DOCUMENT RESUME

ED 289 165 CS 210 841

AUTHOR Matthiessen, Christian; Kasper, Robert
TITLE Representational Issues in Systemic Functional Grammar and Systemic Grammar and Functional Unification Grammar. ISI Reprint Series.
INSTITUTION University of Southern California, Marina del Rey. Information Sciences Inst.
SPONS AGENCY Advanced Research Projects Agency (DOD), Washington, D.C.; Air Force Office of Scientific Research, Washington, D.C.; National Science Foundation, Washington, D.C.
REPORT NO ISI/RS-87-179
PUB DATE May 87
CONTRACT F49620-79-C-0100; IST-8408726; MDA903-81-C-0335
GRANT INST-8408726
NOTE 83p.; Reprint from Proceedings of the International Systemic Workshop (12th, Ann Arbor, MI, August 21-24, 1985).
PUB TYPE Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC04 Plus Postage.
DESCRIPTORS *Artificial Intelligence; Cognitive Processes; Cognitive Psychology; *Computational Linguistics; Computer Uses in Education; Discourse Analysis; English; Language Patterns; *Language Processing; Language Research; Linguistics; *Linguistic Theory; Psycholinguistics; Semantics; Syntax
IDENTIFIERS Functional Linguistics; Functional Unification Grammar; Halliday (M A K); Metalinguistics; Natural Language Processing; Systemic Grammar

ABSTRACT

Consisting of two separate papers, "Representational Issues in Systemic Functional Grammar," by Christian Matthiessen and "Systemic Grammar and Functional Unification Grammar," by Robert Kasper, this document deals with systemic aspects of natural language processing and linguistic theory and with computational applications of M. A. K. Halliday's systemic functional grammar. The first paper is concerned with representation as it is defined by linguistics--how to represent language metalinguistically, or how to talk about talk. Topics covered in the paper include: (1) the lag between theory and representation; (2) dimensions of structure (including a sample grammar and discussion of the structuring, insert, expand, conflate, and preselect operators); (3) modes of meaning and modes of structure, a characterization of representational problems of constituency, prosody, pulse, and interdependency; (4) modes of meaning and modes of choice; and (5) possible solutions. The paper also contains an appendix on representational conventions and a bibliography. The second paper looks closely into how systemic functional grammar can be represented using the framework of functional unification grammar (FUG). Included are descriptions of: the ingredients of FUG (unification, paths, patterns, disjunctions, and their representations); technical details of the representation of systemic grammar in FUG; differences between FUG and systemic grammer; extensions to FUG motivated by systemic functional grammar; solutions to problems in systemic notation (such as discontinuous constituents and feature propagation); and features of a grammatical analysis computer program. References are included. (SKC)

ED289165

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

This document has been reproduced as received from the person or organization originating it.

Minor changes have been made to improve reproduction quality.

Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ISI Reprint Series
ISI/RS-87-179
May 1987

University of Southern California

Christian Matthiessen
Robert Kasper

Representational Issues in Systemic Functional Grammar - and- Systemic Grammar and Functional Unification Grammar

Reprinted from the *Proceedings of the12th International Systemic Workshop in Ann Arbor, MI., August 21-24, 1985.*

CS210841

INFORMATION SCIENCES INSTITUTE

213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

BEST COPY AVAILABLE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT This document is approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/RS-87-179	5. MONITORING ORGANIZATION REPORT NUMBER(S) ---
6a. NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION ---
6c. ADDRESS (City, State, and ZIP Code) 4676 Admiralty Way Marina del Rey, CA 90292	7b. ADDRESS (City, State, and ZIP Code) ---
8a. NAME OF FUNDING / SPONSORING ORGANIZATION National Science Foundation (over) 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER IST-8408726 F49620-84-C-0100 MDA903-81-C-0335
8c. ADDRESS (City, State, and ZIP Code) 1800 G St. NW (over) Washington, DC 20550	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. --- / PROJECT NO. --- / TASK NO. --- / WORK UNIT ACCESSION NO. ---

11. TITLE (Include Security Classification)
A. Representational Issues in Systemic Functional Grammar -and-
B. Systemic Grammar and Functional Unification Grammar (Unclassified)

12. PERSONAL AUTHOR(S)
Matthiessen, Christian; Kasper, Robert

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Research Report	FROM _____ TO _____	1987, May	83

16. SUPPLEMENTARY NOTATION
Reprinted from the *Proceedings of the 12th International Systemic Workshop in Ann Arbor, MI.*, August 21-24, 1985.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
artificial intelligence, functional unification grammar, grammar, linguistics, natural language, Nigel, parsing, syntax, systemic grammar, text generation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A.
Nigel is a large diverse computational grammar for text generation. Its framework is an implementation of Systemic Functional Theory of grammar and it constitutes a context in which the representation of systemic theory can be explored and studied.

This paper surveys the representational devices used in the Nigel grammar and the representational issues that they raise in relation to systemic theory. These issues are diagnosed in the light of the metafunctional differentiation of systemic theory.

(over)

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☑ UNCLASSIFIED/UNLIMITED ☑ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Sheila Coyazo Victor Brown	22b. TELEPHONE (Include Area Code) 213-822-1511 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

8a. (continued)

Air Force Office of Scientific Research
Defense Advanced Research Projects Agency

8c. (continued)

Building 410, Bolling Air Force Base
Washington, DC 20332

1400 Wilson Blvd.
Arlington, VA 22209

19B.

Systemic Functional Grammar (SFG) and Functional Unification Grammar (FUG) are superficially very different approaches to grammatical knowledge, but they share an underlying comparability that runs very deep. FUG shares with systemic descriptions an emphasis on the functions of linguistic objects, and an explicit representation of feature choices. This paper explores how a systemic grammar can be represented in FUG notation, as a step toward creating a grammatical analysis program for English. Because FUG has been developed as a computational tool, expressing a systemic grammar in FUG notation allows new computational techniques to be applied to it Among other benefits, this program will make it possible to study how much the grammatical functions of sentences are recoverable from them. It will also provide a method to test the amount of ambiguity implicit in a systemic description, a topic which has so far been inaccessible. This use of FUG as an alternate representation for SFG may have some additional benefits for both frameworks It provides some solutions to problems in systemic notation which are described by Matthiessen (in this volume). Several extensions to the FUG framework are also suggested by this study

ISI Reprint Series
ISI/RS-87-179
May 1987

University of Southern California

Christian Matthiessen
Robert Kasper

Representational Issues in Systemic Functional Grammar - and- Systemic Grammar and Functional Unification Grammar

Reprinted from the
Proceedings of the12th International Systemic Workshop in Ann Arbor, MI., August 21-24, 1985.

INFORMATION SCIENCES INSTITUTE

213/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

This research was supported in part by National Science Foundation Grant No. IST-8408726, in part by the Air Force Office of Scientific Research Contract No. F49620 79 C 0181, and in part by Defense Advanced Research Projects Agency Contract No. MDA903 81 C 0335. The views and conclusions in this report are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either explicit or implied, of the National Science Foundation, the Air Force Office of Scientific Research, the Defense Advanced Research Projects Agency, the U.S. Government, or any person or agency connected with them.

ISI Reprint Series

This report is one in a series of reprints of articles and papers written by ISI research staff and published in professional journals and conference proceedings. For a complete list of ISI reports, write to

Document Distribution
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695
USA

1 TALK ABOUT TALK[1]

The lexicogrammar of a language represents the semantics of that language, not as one half of a sign represents the other half, but as a whole system of systems represents another system of systems: representation as it is built into the organization of natural languages is the symbolic relation we find or posit between different strata (levels) of linguistic organization. Consequently, this is the kind of representation the linguist is concerned with. But he/she must also be concerned with representation as it is defined by linguistics itself, viz., the linguist's representation of natural language according to some theory of language. The issue here is how to represent language metalinguistically; how to talk about talk.

The two tasks of representation are comparable in many respects. Indeed, we could turn linguistic theory back on itself and use the framework that has been devised for the study of language to understand and talk about metalanguage. For instance, we could try to do systemic metalinguistics in this way. What I will do in this paper is in fact metalinguistics, but it would take too long to try to situate my topic within a systemic metalinguistics here. It will, however, be useful to say something about the context. The central topic is the representation of lexicogrammar.

The general purpose that has prompted the present exploration of representational issues is **text generation**. It is text synthesis and contrasts with text analysis, but both processes can be used to study text. It is the process of creating a text in response to a communicative goal (purpose). The process is specified in a text generation model (system). As a metalinguistic purpose for linguistics, text generation can be compared and contrasted with other purposes such as text understanding, text analysis, the grammatical tagging of a text corpus, literary interpretation, language education, contrastive analysis, and typological linguistics. These purposes are all, of course,

---

[1]In my work on representational issues, I have profited from discussions with a number of people, including Michael Halliday, Bill Mann, Bob Kasper, Robin Fawcett, Peter Fries, Jay Lemke, and John Bateman, and participants in the Systemic Workshop in Ann Arbor, all of whom I am very grateful to. I am also greatly indebted to Susanna Cumming, Bob Kasper, and Lynn Poulton for helpful comments on a draft of this paper. All errors and oversights are my own.

systematically related as studies of language, but they also occur in different contexts and raise different demands, e.g., on the representation of the theory. For example, perspicuity is probably higher on the list of demands on grammatical representation in language education than in text generation.

The computer is a highly useful tool in work on text generation -- not necessary, but useful. It helps the linguist manage large bodies of data, such as a lexico-grammar, and it enables him/her to run actual tests of the model of text generation being built. There are several examples of systemic work on text generation or work using a computational grammar closely related to systemic ideas. In the early 1970s, Davey (1978) used a systemic grammar partly derived from Hudson's work (e.g., Hudson (1971)) in a text generation system. McKeown (1982; 1985) used a Functional Unification Grammar (FUG) in her TEXT system, and Appelt has included a FUG as one of the levels in his generation system (Appelt (1983)). FUG was developed by Martin Kay (see e.g., Kay (1979)) and is closely related to systemic functional grammar; the grammar and its relationships to systemic grammar are discussed in Bob Kasper's paper in this report.

In 1980, work on a large systemic grammar of English for text generation started at the Information Sciences Institute of the University of Southern California. It is called the **Nigel** grammar and is being developed as an integral part of a text generation system. Bob Kasper is currently working on a version of Nigel to be used for parsing. John Bateman has started work on a computational systemic grammar of Japanese, also in the context of text generation (Bateman (1985)). Also, Robin Fawcett is developing a computational systemic grammar of English, implementing his contributions to systemic grammar (cf. Fawcett (1980)). The present discussion is derived from the work on the Nigel grammar. For example, the constraints on realization operators to be discussed are constraints in the Nigel grammar. The framework of the Nigel grammar is intended to be as faithful as is currently possible to Michael Halliday's ideas. My paper is not, however, an introduction to the Nigel grammar; for brief presentations see e.g., Mann & Matthiessen (1985), Matthiessen (1983), and Mann (1983).

What is the context of text generation, particularly in computational work? (For a discussion of related issues, see also Fawcett's paper in this volume.) The context can be characterized with the help of systemic theory, using field (the social action and subject matter, what's going on), tenor (the social relationship, who's taking part), and mode (how are the meanings exchanged); see e.g., Halliday & Hasan (1985).

In the field, processes are foregrounded. There must be dynamic accounts of traversing the system network, activating realization statements, etc. in addition to static accounts, in the text generation model; the two perspectives have been discussed by Martin (1985). Most descriptive work has been focused on the system network at rest and on structure as a static product of structure building. However, there are often two interpretations, a dynamic one and a static one. For example, systems can be read as instructions to choose under certain conditions (dynamic) or as statements about the availability of options (static).

In the tenor, explicitness and formality are emphasized since the user of the text generation model is a computer and not a knowledgeable linguist as in a lot of descriptive work. When there are gaps in the account, the computer cannot fill them in. It cannot clarify ambiguous statements.

The mode has to be algebraic rather than graphic; it has to be a mode that can be manipulated in computer programs. In descriptive work, the graphic mode is very valuable and has usually been favoured. Systems and system networks are represented by graphic diagrams; and structures are represented by box diagrams.

Advances have been made in the work on Nigel in all three areas of the context so that more demands can be met. For example, more is known about the process of traversing a system network and about constituency ordering. However, the focus of this paper will be on representational issues that have been discovered and remain and not on these advances. Although I will discuss some approaches to solutions, this is essentially a paper about problems and not about solutions. Whenever we consider solutions, there is a high demand on them: the representation should be as natural and

as iconic as possible with respect to the theory. Functional felicity is more important than generative power.

Although I will try to mention the various aspects of systemic work the paper addresses, the paper presupposes some familiarity both with systemic theory and with systemic descriptions of English.

2 THEORY AND REPRESENTATION: LAG

There seems to be a general tendency for theory to precede the representation of it, which is, of course, not surprising. There is a lag between theory and representation, but the lag depends on the nature of the representation. Systemic linguistics has always allowed for a cline in the explicitness of the representation: diagrams and algebraic statements that can be interpreted formally coexist. This cline has proved very useful, since different contexts place different demands on the representation. Aspects of the theory can be represented diagrammatically, although we do not yet know how to give them a representation explicit enough to be implemented computationally. (Cf. Martinet's distinction between visualization and formalization in Parret (1974).) For example, as we will see in Section 4.3, there is a graphic representation of the kind of structure generated by the textual metafunction, "pulsive" or "periodic" structure, but we do not yet know how to represent it in a formal algebraic mode.

Linear recursion. An example of explicit representation lagging behind theory comes from the logical component of grammar. The theory of linear recursion had been developed by the mid 1960s (cf. for example Halliday (1965); Hudson (1971: 60-63)). A recursive system can be represented in an informal graphic way; see Figure 1. (Conventions are given in the Appendix. There is a system with two options represented by features, simplex vs. complex. If the feature 'complex' is chosen in the system 'simplex/complex', the entry condition to the system 'a/b' is satisfied, but there is also a "recursive" loop back to the system 'simplex/complex'. Associated with the feature complex is a realization statement (x -> y, given in the box under the feature complex), specifying a structure fragment. The meaning of it will be discussed in Section 4.4.)

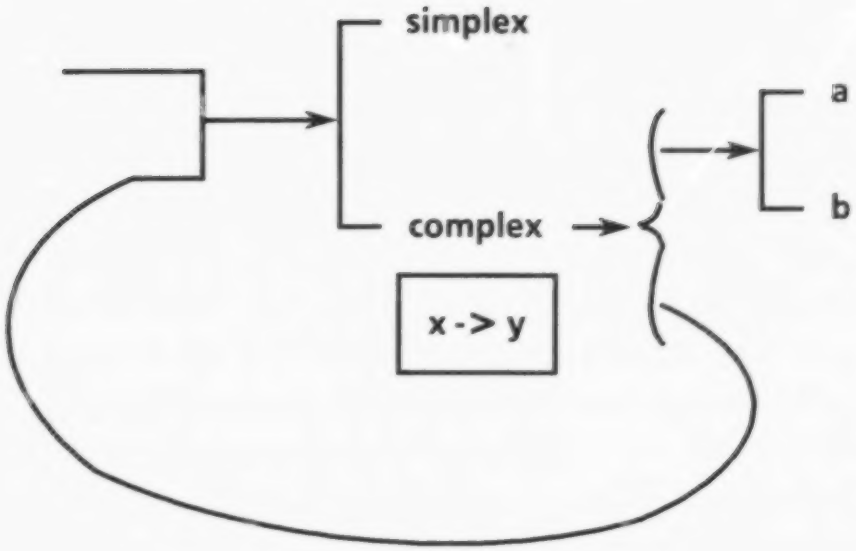


Figure 1: Informal graphic representation of recursive system

However, although the theory of linear recursion had been developed, the problem of the explicit representation of it had not been solved. Henrici worked with an explicit version of systemic representation. Comparing embedding recursion (recursion through rankshift) with linear recursion (iteration), he notes that the latter is "rather more troublesome" (Henrici (1966)). One problem is that the recursive system (i.e., a system with a loop back to its disjunctive entry condition) "would not be a conventional system, in that at least one of the terms in the system would also act as an entry condition to the system, contrary to the partial-ordering imposed on ordinary systems". In other words, the explicit interpretation of the system representation as a partial ordering brought out the issue of what the recursive system means. We have had similar experience in our work on the Nigel grammar.

In the case of linear recursion, the lag between theory and explicit representation is

still very evident. Davey did not implement recursive systems in his computational generation grammar; he writes (1978: 81): "It is clear that systemic functional grammar cannot produce coordinate items without some extension of its purely classificatory apparatus, but we do not feel confident that the right extension has been found." Similarly, McCord (1975: 211) discusses a modification of conventions to handle linear recursion. There is no generally accepted explicit interpretation of it for a formal representation, neither for the recursive system (see Section 5.1) nor for the structural output (see Section 4.4).

Box diagrams and realization statements. The representation of structure in general is another example of the lag between theory and explicit representation, as we will see. There are various structural relationships (e.g., discontinuity, which I will discuss later) that do not pose a problem for the informal box diagram representation but prove to be a problem for explicit realization statements.

The box diagram is like a tree diagram in that it represents a product rather than a process, the product of a structure specification (realization statement), but it is used instead of the tree diagram because it facilitates the representation of layering (see Figure 9). Examples of box diagrams can be found in Figures 3, 10 and 15.

The representational problems I will identify and discuss are usually problems for a representation that is more explicit than a box diagram representation. I will make reference to Martin Kay's Functional Unification Grammar (FUG) and Bob Kasper's paper in this report where this kind of grammar is presented and discussed as a source of possible solutions to the problems: FUG was designed to be explicit and formal enough for the computational tasks of generation and parsing.

3 DIMENSIONS OF STRUCTURE

All parts of a systemic grammar are ultimately interconnected. The grammar is a network of systems of options; each option may have one or more realization statements associated with it, serving to specify a structure fragment or to predetermine the choice of an option as the grammar is re-entered. Given this interconnectedness, we could

choose various starting points in a discussion of representational issues, for example the representation of the network of systems or the representation of structure. We could discuss the issues in terms of the grammar as a system at rest (the potential) or of the grammar as a process (the potential actualized). We could choose one of these various angles on lexicogrammar and still hope to arrive at a diagnosis of representational issues that is reasonably general.

In this section, I shall unravel the grammar by means of the various realization operators used in Nigel: conflate, insert, order, expand, and preselect. They serve to specify the structural output of the grammar; the representational issues will consequently be exemplified first in terms of the specification of structure. There are various reasons for starting with the realization operators. One is that they have been discussed and explored less than the system network representation. Another is that they are more restricted in representational applicability than the system network representation (as we will see) and will serve to identify representational issues more clearly.

3.1 Three dimensions of structure

Structure is a composite object or process in which each fragment is specified by the realization operators. For the discussion of the realization operators, it is useful to think of structure as organized in terms of three dimensions (cf. Halliday 1961; Matthiessen 1985). The dimensions are manifested both in the system's potential for creating structures and in examples instantiating this potential. For each dimension we find one or more realization operators:

1. Structuring: insert, order, expand
2. Metafunctional layering: conflate
3. Rank: preselect

Structuring and metafunctional layering define the structure of a unit such as a clause, a nominal group, or a verbal group as a configuration of functions. The functions are realized by units of the rank below, through preselection. (There is a

distinction between a structure, a configuration of functions, and a syntagm, a sequence of classes such as nominal group and verbal group: see Halliday (1966). To keep the discussion manageable, I will not bring in the distinction, but will merely note here that it exists and could be used to address certain representational problems. The realization operator Preselect establishes the relation between a function in the function structure of a unit at one rank and a grammatical class represented as a feature at the rank below.) The realization operators to be discussed are as follows:

Name of operator		Description
(Insert Subject)	+Subject	Function inserted as constituent of the structure of the unit being specified
(Order Subject Finite)	Subject^Finite	One function ordered to precede another function
(Expand Mood Finite)	Mood (Finite)	One function is expanded to have another funciton as constituent
(Conflate Subject Agent)	Subject/Agent	One function is conflated with another function to form part of the same constituent function bundle
(Preselect Subject singular)	Subject: singular	A function is preselected for a feature; the realization of the function is constrained to display that feature

3.2 Sample grammar

Before discussing the (Nigel) realization operators, I will present a fragment of the Nigel grammar so that I can refer to it later for illustrations: see Figure 2. The network fragment consists of systems belonging to MOOD, an interpersonal region in the clause grammar, to TRANSITIVITY, an experiential region in the clause grammar, and to VOICE, a textual region:

MOOD systems: MOOD TYPE, INDIC[ATIVE] MOOD PERSON, INDIC[ATIVE] TYPE, INTER[ACTANT MOOD PERSON] TYPE, and [DECLARATIVE] SUBJ[ECT] PRES[UMPTION]),

TRANSITIVITY systems: AGENCY and AGENTIVITY, and

VOICE system: VOICE.

Since it is only a partial fragment, the grammar in Figure 2 cannot be used to generate complete clause structures. Graphic conventions are given in the Appendix. For present purposes, the systems MOOD TYPE and AGENCY can be assumed to be simultaneous, i.e. reachable from the same entry condition, viz., the feature clause.

Before the part of the grammar specified in the diagram above is entered, the following has occurred: the functions Predicator, Medium, Process, and Finite have been inserted; Predicator and Process have been conflated; Process has been preselected to be a verbal group; and Medium has been preselected to be a nominal group. Traversing the mood part of the network, we can choose the following path:

System: feature chosen	Realization statement
MOOD TYPE: indicative	+Subject, Subject: nominative, Mood(Subject)
INDICATIVE TYPE: declarative	Subject ^ Finite

INDICATIVE MOOD PERSON:

interactant

INTERACTANT TYPE: speaker Subject: speaker, Finite: speaker

SUBJECT PERSON: explicit

Traversing the two transitivity systems AGENCY and AGENTIVITY, we can choose

AGENCY: effective Process: effective

AGENTIVITY: agentive, +Agent, Agent: ngp

and in the VOICE system

VOICE: operative Subject/Agent

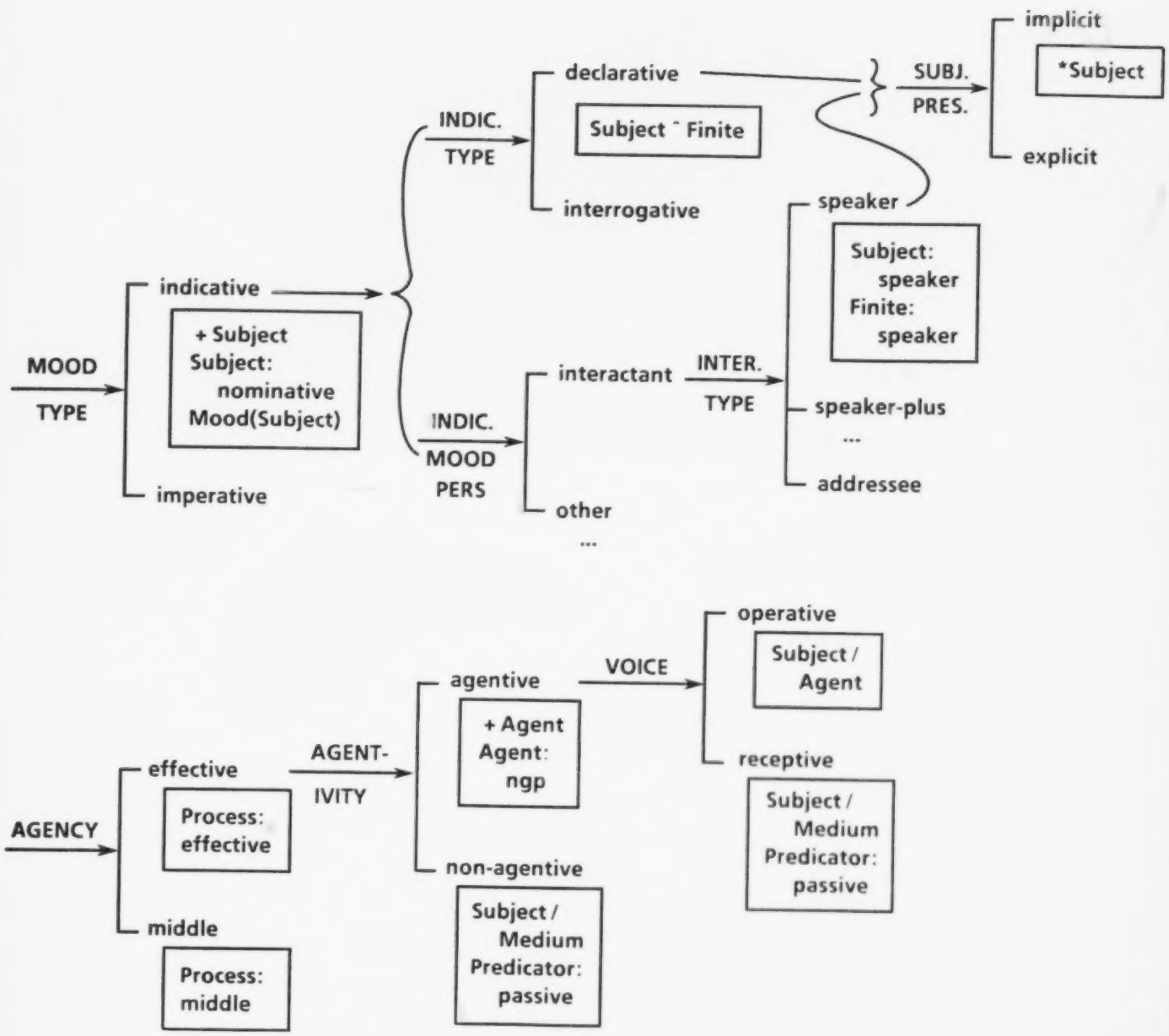
The structure fragment that results from the application of the realization statements is given in Figure 3.

The structure above is a partial specification of examples like *I have dissolved the parliament*, *I have frightened my poor friend*, and *I will open the gates.* After this brief example of the system network, realization statement associated with features in it, and the function structure resulting from the application of realization statements, I will now turn to the first dimension mentioned in Section 3.1.

3.3 Structuring

Structuring is the dimension that defines syntagmatic organization within one rank and within one (metafunctional) layer of structure. Grammatical regions make structural contributions such as transitivity structure (e.g., Agent Process Medium: see Figure 3), mood structure (e.g., Subject Finite: see Figure 3), theme structure (e.g., Theme Rheme), tense structure, and modification structure.

Figure 2: Fragment of clause grammar



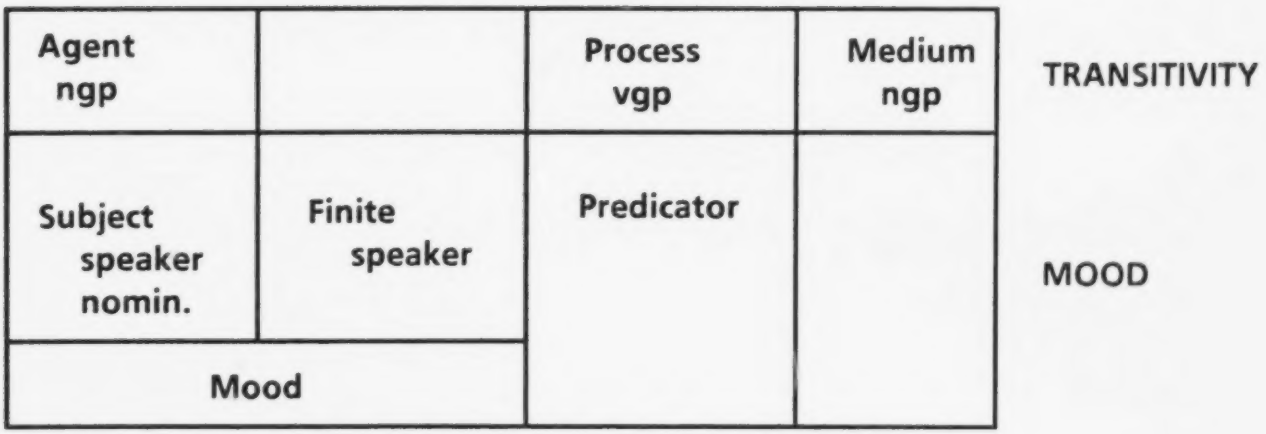


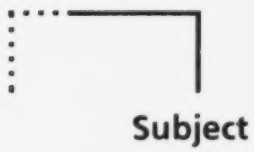
Figure 3: Fragment of function structure

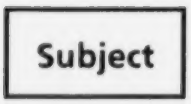
The realization operators used to specify structuring are **Insert, Order,** and **Expand.** Their use will be illustrated with examples from the MOOD region of English grammar. The presentation of each operator is organized into a characterization of it, the result of its application, a specification of some constraints on it, and representational problems.

3.3.1 Insert

Characterization. The **Insert** operator is used with a grammatical function such as Subject, Agent, or Theme as its operand, e.g., (Insert Subject) or +Subject for short. It specifies the presence of the function in (adds the function to) the function structure being generated as a constituent of that structure. For example, in the specification of a clause, we may associate the realization statement +Subject with the grammatical feature indicative as in the grammar fragment above. This means that the function Subject is present in the structure of a clause as a constituent of the clause structure.

Result. The result of the application of +Subject is simply a constituent branch and node in a tree diagram or a constituent box in a box diagram; see Figure 4.





(i) Tree diagram *(ii) Box diagram*

Figure 4: +Subject -- structural result in diagram

Constraints. I will mention two important constraints on the Insert operator, one having to do with the status of an inserted function and one with its constituency:

1. When an inserted function is declared to be present, "present" means 'actually present' and not 'potentially present' or 'present in the default case'. Once inserted, a function is, and remains, actually inserted.

2. The fact that Subject is a constituent of clause structure means that it is available to other structure specifications within the clause but not outside the clause. It also means that it is not a constituent of any other unit, e.g., arankshifted (embedded) constituent clause. Furthermore, a clause cannot insert a function into e.g., a verbal group or a nominal group.

Problem: implicit functions. Both constraints are well motivated in principle, but there are situations where they lead to some complex conditions in the grammar. Consider the first constraint.

How do we deal with implicit functions, e.g., implicit Subjects? The function Subject is explicit, actually present, in almost all indicative clauses. Consequently, it is tempting to associate the realization statment +Subject with the feature indicative. If we do (as in system MOOD TYPE of the sample grammar in Figure 2), we will get

Subjects even where we do not want them, where they are implicit, as in *[I] haven't had dinner yet.*

There is no formal problem with adding to the conditions under which the statement +Subject is applied so that Subject does not get inserted where it should be left implicit, but the added conditions do complicate the grammar a bit. The insertion of Subject can be stated in a gate whose disjunctive entry condition specifies under what conditions a clause may have an explicit Subject. (A gate has an entry condition like a regular system, but only one output feature/term/option.) The output is simply one feature, explicit subject, with which the realization statement +Subject is associated. This is illustrated for the sample grammar in the diagram below; cf. the gate for Subject insertion in Matthiessen (1985: Figure 9). (The grouping of the features of the entry condition is not formally significant.)

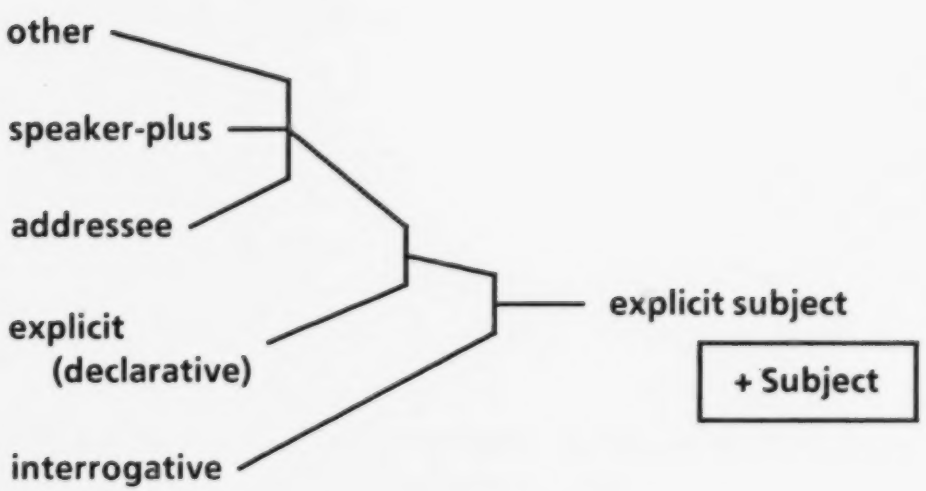


Figure 5: Gate for inserting Subject

The complication is increased when we add the conditionality for functions such as Agent that may get conflated with Subject; see Section 3.4.

One solution. There is one way of dealing with the problem which involves a re-interpretation of the Insert operator. Instead of defining its result as 'actually present', we can define it as 'potentially present', i.e. 'present in the *default* case, unless the function is declared not to be present'. We can then add a preventive operator, call it *, to be used to specify absence. Assume that Subject has been inserted, + Subject (MOOD TYPE: indicative). If the grammar states *Subject, the Subject, although potentially present, is not actually present (system DECLARATIVE SUBJECT PRESUMPTION: implicit). There would be two possible cases if a function has been inserted:

```
+ Subject  'potential presence'

        [1]  * Subject 'not actually present'

        [11]   ---     'actually present, by default'
```

Potential presence can be compared with presence of a function in Fawcett's starting structure (described in Fawcett (1980)).

3.3.2 Order

Characterization. The operator **Order** is applied to two functions, e.g., (Order Subject Finite) or Subject ^ Finite. It introduces a precedence relation between the two functions in realization sequence. There are two varieties of this operator, one that only specifies the precedence relation and one that also specifies contiguity so that no other constituent can intervene. I shall not distinguish them here. (I shall also omit OrderAtEnd and OrderAtFront; cf. Mann & Matthiessen (1984) and Matthiessen (1985).)

The realization operators that relate two functions or a function and a feature specify relations that can be classified in terms of Halliday's theory of relational transitivity (Halliday 1985). I will not elaborate on this typology of realization operators here, but

will indicate what relations they specify as we meet them in the presentation. For example, the Order operator specifies a circumstantial (enhancing) relation, e.g., Subject precedes Finite.

Result. The result of the application of Finite ^ Subject is represented diagrammatically as follows:

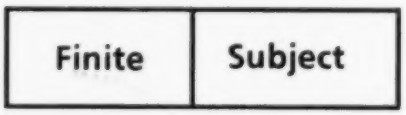


Figure 6: Finite ^ Subject -- structural result in diagram

In a box diagram, we usually cannot tell whether the sequence is significant, i.e. introduced by a sequencing statement like (Order Finite Subject), or not.

Constraints. I will mention three constraints on the Order operator, all of which have to do with ordering and constituency status:

1. The Order operator specifies actual sequence and not potential sequence; it does not specify a default sequence that could be overridden.

2. Only constituents of the unit whose structure is being specified can be ordered in relation to one another. For example, the clause constituent Subject can be ordered with respect to the clause constituent Finite but not with respect to a Qualifier from a nominal group structure.

3. Once a functional constituent such as Subject has been ordered, all the constituents of the unit realizing that function (Subject) have to obey the ordering specification.

Although there is no inter-rank ordering, there is no constraint that disallows inter-layer ordering, i.e. ordering statements involving functions from different layers of structure. For example, there is no formal problem with an ordering statement such as

Subject ^ Medium with one mood function and one transitivity function. However, there has not been any reason to use ordering statements of this kind so far in Nigel's grammar of English.

There is a mechanism for ordering other than the Order operator: default ordering lists. They list functions in the sequence in which they will appear unless another ordering has been specified explicitly by means of the Order operator. The first constraint says that the Order operator always specifies the actual sequence. In contrast, the lists specify potential (default) sequence. A default ordering list in Nigel is comparable to Fawcett's starting structure as an ordering statement.

The next two constraints are appropriate most of the time, but there are certain situations where they are too strict, viz., theme and information prominence. These situations typically involve discontinuity; one case will be discussed next.

Problem: discontinuity. Consider the example *They called the meeting off* from Halliday (1985), with the process *call ... off*. Halliday gives it the following transitivity structure in box diagram representation (the example is not covered by the grammar fragment in Figure 2).

Although the box diagram looks different from those that have been used up to now, there is no problem in drawing it. In other words, the discontinuity can be represented in the box diagram. However, there is a problem for the formally interpreted realization operators described above. None of these operators can serve to specify the structure represented by this box diagram.

For example, we cannot state Process ^ Medium ^ Process because that is contradictory, claiming that Process both precedes and follows Medium. Two approaches to this kind of problem have been used in the literature.

1. Systemic grammarians have sometimes used a **subscript** to indicate two parts of a **split** function: Process_1 ^ Medium ^ Process_2 where the first part is realized by *called* and the second part is realized by *off*. However, this device does not make explicit what the two parts are; the identification of them is left to the knowledgeable grammarian.

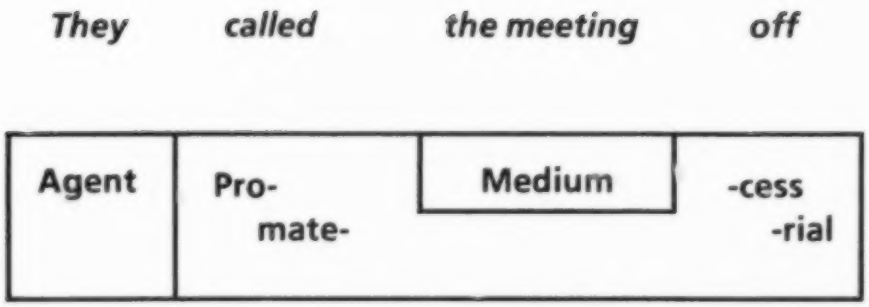


Figure 7: Transitivity structure of discontinuous example

2. Alternatively, we can introduce a new realization operator. For example, Huddleston works with the notion of **inclusion**: one function can be included within another function. Thus, we would have (Include Medium Process), usually Process <Medium>. The problem with this strategy is the same as with the above-mentioned splitting strategy. If we only specify that one function (Medium) is to be included within another function (Process), this does not provide enough information to assign the first function a place within the second function. This problem does not invalidate the use of inclusion, of course, but it indicates that inclusion alone may not be enough, since more information is needed for an unambiguous specification of ordering. We could take inclusion to mean that the function included within another function is not given a unique ordering until the cycle through the grammar when the second function is realized. This method would need more communication between ranks than is presently used in Nigel.)

In spite of the problem with the analysis of *called ... off* as a discontinuous realization of Process, we do not want to give it up. The sequential discontinuity is matched by lexical continuity: 'call off' is one lexical item; it is not semantically decomposable.

I shall return to the problem of discontinuity after discussing the second dimension of organization, i.e., metafunctional layering, in Section 3.4.

3.3.3 Expand

Characterization. The operator Expand is applied to one function to be expanded and one or more expanding functions, e.g., (Expand Mood Subject) or Mood (Subject), Mood (Subject, Finite), and so on. It creates a constituency relation between the expanded function and the expanding function(s). Thus, Mood (Subject) means that Subject is one of the constituents of the Mood constituent.

When the Insert operator is applied to a function, the function is inserted as a constituent of the unit being generated. (It does not have to be an immediate constituent.) For example, the statement +Subject inserts Subject as a clause constituent. The Expand operator is used when a function is not an *immediate* constituent of a unit (which it is by default) but rather an immediate constituent of a function in that unit. For example, Mood (Subject) specifies that Subject is an immediate constituent of Mood rather of the clause. (In principle, indefinitely long constituency chains could be created; in practice, there has never been any need for a situation more complex that the one illustrated by Mood (Subject), where Mood is an immediate constituent of the clause and Subject is a constituent of the Mood constituent.)

The Expand operator specifies a possessive (extending) relation of the part-whole kind, e.g., Subject is part of Mood. (To bring it up to date with Halliday's terms for the clause complex relations, we should perhaps call the operator Extend.)

Result. Figure 8 diagrams the structural result of the application of Expand.

Constraints. There are two constraints, the second of which anticipates another realization operator, Preselect, which is described in detail in Section 3.5:

1. Expansion is actual expansion, not potential expansion.

2. A function expanded by another function cannot be preselected for a feature. Only functions that are immediately realized by a grammatical class can be preselected. There is no mechanism for spreading or distributing preselected features from an expanded function to its expanding functions.

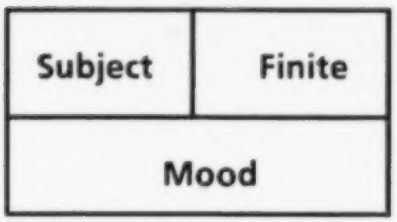


Figure 8: Mood (Subject, Finite) -- structural result in diagram

One example of the consequences of the second constraint will be mentioned in Section 4.2. Expand is not used often. In Nigel, its main uses are in mood structure reflecting prosodic organization (see Section 4.2) and in theme structure reflecting thematic prominence (see Section 4.3). In the second case, it crosses metafunctional layering (see Section 3.4), as in Theme (Locative), where Theme is textual and Locative is experiential.

3.4 Metafunctional layering

Metafunctional layering is the dimension that defines the organization across layers of syntagmatic structuring such as transitivity structure, mood structure, and theme structure; it represents the reconciliation of these various metafunctional contributions.

Characterization. The realization operator used to specify metafunctional layering is **Conflate**. It is applied to two grammatical functions, e.g., (Conflate Subject Agent) or Subject/Agent (sometimes the equal sign has been used, Subject = Agent). It means that they describe the same constituent. Conflate specifies an identifying relation: Subject is Agent; Agent is Subject.

Result. The structural result of conflation is called a function bundle or fundle. It is represented by a column in a box diagram; see Figure 9 (There is no commonly used

tree diagram representation of layering, although there is a tree-like diagram in Relational Grammar where the so-called strata are comparable to metafunctional layers.)

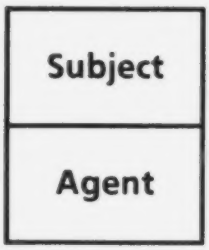


Figure 9: Subject / Agent -- structural result in box diagram

Constraints. I will discuss three constraints and their associated problems.

1. Conflation is actual conflation, not potential conflation. Both functions in a conflation statement have to be inserted at some point during the same pass through the grammar as the conflation. If one or both functions do not get inserted, there is an error in the grammar.

2. Only functions from the structural specifications of the same unit can be conflated; there is no inter-rank conflation. For example, Subject (from the mood structure of the clause) and Agent (from the transitivity structure of the clause) can be conflated but Subject and Minirange (from a prepositional phrase structure) cannot.

3. The functions conflated in a function bundle such as Subject/Agent always describe the same constituent. It is not possible to apply Conflate to functions and maintain different constituents for the different functions. This constraint is related to the discontinuity problem that I introduced in Section 3.3; I will return to the problem later.

Problem: implicit functions (continued from Section 3.3.1). What happens if a function that has been inserted, say Agent, gets conflated with a function that has not been inserted, say Subject? This situation was addressed above in the first constraint:

if Subject gets inserted after the conflation, there is no problem. As long as both functions in a conflation statement get inserted at some point, the conflation is valid. However, if one of the two functions does not get inserted, this is considered a grammatical error. This is quite reasonable in most cases, but we have to consider how the constraint affects implicit functions, the problem introduced in Section 3.3.1.

Assume that the transitivity of a clause is such that the function Agent gets inserted (system AGENTIVITY: agentive in the grammar fragment above in Figure 2). Assume further that the clause is operative (active; the VOICE system), resulting in a conflation of Agent and Subject: Agent/Subject. Now, if mood choices lead to an explicit Subject, there is no problem: Subject will be inserted as was Agent. But if mood choices lead to an implicit Subject (system DECLARATIVE SUBJECT PRESUMPTION: implicit), there is a problem: Agent is inserted but Subject is declared to be implicit and hence absent ("*Subject" in the grammar fragment above), resulting in a conflation between an inserted function and an absent function. How can the presence of Agent be blocked?

Solutions. (i) One solution is simply to add conditions for the insertion of Agent so that the function does not get inserted unless Subject is explicit in an operative clause. This approach is used in Nigel and poses no formal problems. There is a gate, a "system" with only one output feature, for inserting Agent with a complex entry condition (a conjunction of the features agentive and explicit subject, from the sample grammar and the gate for inserting Subject, respectively). The complexity of the entry condition is a drawback, of course. It would arise for each function that may be conflated with Subject.

(ii) Another solution is to rely on the distinction between potential presence and actual presence discussed in Section 3.3.1. When the statement +Agent is applied, the function Agent is potentially present. If it gets conflated with another function, Subject, that has been inserted as potentially present, both functions will be actually present unless there is a statement to the contrary. There could be a preventive statement, specifying the absence of Subject (as in the system DECLARATIVE SUBJECT PRESUMPTION

in the sample grammar): *Subject, which would cause the conflation Agent/*Subject to be absent (implicit) as a whole. Or alternatively, Subject may never get inserted, in which case a convention would prevent the potential presence of Agent from becoming actual presence.

These conventions are not ad hoc, but functionally motivated: The presence of an element of structure or a bundle of functions depends on more than one metafunction; it is a layered object and unless all the layers are present, the object itself won't be present.

Problem: conflation outside unit. The second constraint on conflation is usually desirable. In fact, it reflects an important aspect of the functional organization of a grammar: the units of the grammar are units because they unite particular functional contributions, e.g., mood, transitivity and theme for the clause. However, there are some cases, such as the Minirange of a prepositional phrase realizing a locative circumstance of a clause *(This bed has not been slept in)*, where the constraint is problematic.

Problem: discontinuity (continued from Section 3.3.2). I left the example *They called the meeting off* without a solution in Section 3.3.2. I noted that it exemplifies both continuity and discontinuity in the process *call ... off*. Lexically, it is continuous; that is, it is one lexical item. In terms of sequence, it is discontinuous.

Towards a solution of the discontinuity problem. Halliday (1985) provides a solution to the problem of representing examples with discontinuously realized phrasal verbs. It relies on the metafunctional layering, so it could not be stated only in terms of one dimension of structuring (Section 3.3).

Halliday's account involves both transitivity structure and mood structure. The two interpretations for *They called the meeting off* are:

1. Transitivity: one Process (of class 'material') *call off*. The lexical specification can be stated in terms of Process: Process: 'material'. (In other words, Process is preselected for the lexical feature 'material'.)

2. Mood: one Predicator *call* and one Adjunct *off*. The sequence can be stated in terms of Predicator and Adjunct: Predicator ^ Complement ^ Adjunct.

Each interpretation alone captures only part of the facts, but we get the full picture, when they are brought together. The mood structure is also used to state the inflectional form of the verb. The function Finite is constrained to be 'past', and since the clause is declarative and simple past, Finite is conflated with Predicator. The box diagram for the example is given in Figure 10 (see Halliday (1985)).

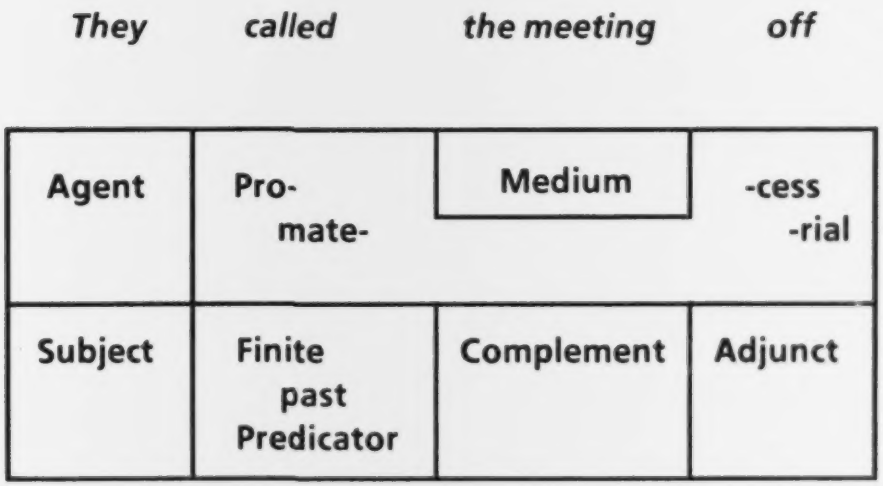


Figure 10: Two layers of structuring for discontinuous example

Halliday's analysis represented in the box diagram avoids the earlier problems discussed in Section 3.3.2: since there is no discontinuity in the sequence Predicator ^ Complement ^ Adjunct, there is no need to design a new inclusion operator and there is no need to split the Process. (The problems with these two solutions were discussed above.) Since we retain an unsplit Process function, there is no problem in constraining it to be one lexical item. Finally, since Predicator and Adjunct are kept distinct, there

is no problem in conflating Finite and Predicator, thus constraining that part of the phrasal verb to be 'past' in our example (i.e. *called off* rather than *call offed*).

There is a problem, however, when we move from the box diagram representation to look for a more formally interpretable realization statement. The problem is the Conflate operator itself. If we conflate Process and Predicator and then also Process and Adjunct to get the column alignment in the box diagram, the result is just one constituent, Process/Predicator/Adjunct. Instead of this conflation, what we need to do is to conflate Predicator and Adjunct with Process without conflating Predicator and Adjunct with one another. We need to be able to state (Predicator & Adjunct)/Process.

Although this kind of conflation does not exist as a Nigel realization operator now, something like it is supported in Functional Unification Grammar: Kasper's paper in this report presents the solution in those terms.

3.5 Rank

The third dimension of organization to be discussed is rank. Structuring and layering combine to form the function structure of one unit such as the clause. Rank organizes the grammar into a scale of units: clause - group/phrase - word - morpheme.

Characterization. The realization operator that handles communication between ranks is **Preselect.** The Preselect operator is applied to a function and a feature. For example, Subject may be preselected to be singular: (Preselect Subject singular) or Subject: singular. This means that the feature singular will be chosen (in the NUMBER system) when the grammar is re-entered to realize the Subject function bundle.

Result. The result of the application of the operator Preselect is like a two-cell tagmeme with a slot (the function) and a filler (the feature). It is represented simply by listing the preselected features under the function they have been associated with. For example, Subject: singular is recorded in the following way; see Figure 11.

Constraints. I will mention two constraints. The first is a familiar one, having to

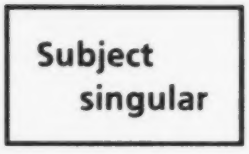


Figure 11: Subject: singular -- structural result in box diagram

do with the actuality of the result of the application of the operator. The second makes reference to the rank scale.

1. Preselection is actual preselection. Once a feature has been associated with a function through preselection it has to be chosen when the function is realized and the system network is traversed. This constraint can be a bit awkward, since it would sometimes be useful to be able to state that a feature is to be selected if its system is entered without forcing the system to be entered. For example, it would be useful to be able to state Subject: nominative for all Subjects, although the feature nominative will only actually be chosen if the Subject is pronominal.

2. In principle, the operator works downwards in terms of the rank scale, from clause to group/phrase, from group/phrase to word, and from word to morpheme. There may also be rankshift, e.g., preselection from clause to clause as group/phrase or from group to clause as word. Preselection cannot be stated within the same rank unless there is rankshift. For example, it is not possible to use Preselect to state a preselection relation between a nominal group and a verbal group when both realize clause functions such as Subject and Finite/Predicator. The constraint can also be phrased in terms of constituency. Preselection works downwards in constituency from a whole to one of its parts; it does not work across from one part to another. The control of selection exercised by Preselect is top-down control.

For example, if we want to use Preselect to ensure agreement between Subject and Finite, we have to set up clause systems for SUBJECT PERSON (or perhaps more accurately MOOD PERSON as in the sample grammar in Figure 2, cf. Section 4.2). If the grammar contains a system for INTERACTANT SUBJECT where one of the features is addressee subject, we can associate the two realization statements "Subject:

second person" and "Finite: second person" to ensure agreemen in examples such as you are. This strategy works up to a point, but there are problems, which are discussed in the following illustration of representational problems.

Problem: agreement. As we have seen, we can achieve agreement by preselecting both Subject and Finite. When Subject is preselected to be plural, i.e., Subject: plural, this feature can be instantiated in the nominal group in at least these four different ways. We have for example:

additive plurality: The title and the purport are ...

plurality in disjunction: The title or subtitles are ...

multiplicative plurality: The titles are ...

collective plurality: The crew are ...

The main problem is one of anticipation. Although these examples are all plural, the nominal groups are grammatically different. For example, the first is plural because the nominal group is an additive complex while the third is a plural simple nominal group. From a grammatical point of view, it is thus hard to anticipate how plurality will be achieved in the nominal group.

It is important to emphasize that the "agreement" between Subject and Finite is a clause phenomenon. Its existence and characteristics are interpretable in terms of the mood organization of the clause, Mood (Subject, Finite) -- cf. Section 4.2. However, the fact that its existence is a clause phenomenon does not entail that the various person and number values necessarily have to be anticipated in the clause.

Towards a solution of the agreement problem. Instead of trying to anticipate the different kinds of nominal group plurality in the clause, it seems preferable to adopt a solution like the following:

1. Specify in a realization statement in the clause that Subject and Finite are the same in person and number, but do not anticipate the various possible values.

2. Let the values be determined across constituents so that the nominal group choices of person and number values (Subject) determine verbal group choices (Finite).

This possible solution seems like an obvious approach. The point here is that it requires new "realization machinery". For the first step, we need to be able to state the identity, say:

```
Finite:  [PERSON = PERSON (Subject)]
```

which means that the PERSON system in the verbal group realizing Finite has the same value as the PERSON system in the nominal group realizing Subject. For the second step, we need to allow the choice of a feature in one system to be made according to the choice of a feature in another system. In other words, we have to allow the method of choosing -- not the particular feature value -- to be predetermined in this case.

This procedure does not exist in Nigel at present. However, something similar is supported by Functional Unification Grammar; see Kasper's paper in this report.

Clearly, there are other possible solutions to the "agreement problem". For instance, instead of having the grammar ensure agreement through some kind of realization statement, we can rely on semantic/conceptual consistency and make the person-number choices twice -- once for the Subject nominal group and once for the Finite constituent. Agreement will then obtain simply because the two sets of choices will be guided by the same information. For example, the referent of the Subject of *King Arthur and his knights are valiant*, 'King Arthur and his knights', will determine both the person and number of the nominal group complex *King Arthur and his knights* and the form of the finite verb are. This is the kind of approach to "agreement" used in Nigel at present.

Whatever solution is adopted to solve the agreement problem, there is a high value to the kind of approach to inter-rank communication supported by FUG and discussed by Kasper. One reason for this is that agreement is one specific instance of a more general phenomenon, viz., a different kind of structuring (prosodic structuring) that is difficult to accommodate with the present set of (Nigel's) realization operators.

3.6 Summary of constraints on structuring operators

The following is a review of the most important restrictions on the realization operators Insert, Expand, and Order:

1. **Actual**: The application of an operator leads to an actual result, not a potential one. There is no way of stating 'potentially inserted/ordered/expanded but not actually [in this case]'.

2. **Domain**: The operators only operate within the domain of the structure of one unit (such as clause structure, nominal group structure, etc.) and within the rank of that unit (clause rank, group rank, etc.).

3. **Compact functions**: Furthermore, the operators operate on functions as discrete objects, without presupposing any internal organization of these functions; there is no overlap, fusion, or partial merger of functions. In particular, there is no realization operator "Include" for inclusion statements.

4. **Complex operands**: Except for the operators Expand and Preselect, the realization operators do not allow complex operands. For example, only Process / Predicator is a possible realization statement, Process / (Predicator & Adjunct) is not.

Up to now, I have only examined a few representational problems, viz., implicit functions, discontinuity, and agreement, and I will now turn to a more general characterization of representational problems.

4 MODES OF MEANING AND MODES OF STRUCTURE: CHARACTERIZATION OF REPRESENTATIONAL PROBLEMS

Pike (1959) and Halliday (1979) have pointed to the existence of different modes of linguistic organization. Pike suggests that we can look at a linguistic phenomenon as field, particle or wave. These different perspectives are complementary and bring out different properties.

Halliday's point is different from Pike's but ultimately related (see Halliday 1979). He identifies four modes of meaning, and suggests that four different modes of grammatical organization correspond to these four different semantic modes. The modes correlate with the different metafunctions:

experiential -- constituency

interpersonal -- prosody

textual -- pulse

logical -- interdependency (chain)

These modes represent four different principles of structuring, i.e. prosodic structuring, and so on. In a function structure, two or more modes may exist in parallel through metafunctional layering. This happens in the clause, which is characterized by layers of transitivity constituency, mood prosody, and theme pulse.

The tools used in representing grammatical structure are well suited only to the constituency mode of organization, and this gives us a basis for looking at representational problems as related in particular to the other modes.

4.1 Constituency

I will start with the mode of organization our grammar is good at, viz., constituency. Constituency is the grammar's mode for organizing experiential meaning: our experience is organized grammatically in constituency terms. For example, a process composite/configuration is decomposed into constituent parts such as the process itself, the participants involved in it, and the attendant cirumstances (transitivity). A participant is usually an object, and is further decomposed grammatically as a thing and classifiers, epithets, and qualifiers (modification).

Our common notion of grammatical structure fits this mode of organization quite well; we speak of grammatical constituents and draw tree diagrams. More specifically, the

realization operators discussed in Section 3 are constituency oriented. The structuring operators (Insert, Expand, and Order) create constituency, while the layering and rank operators rely on constituency.

1. *Insert* specifies the presence of a function as a constituent of a grammatical unit.

2. *Expand* explicitly creates a constituency relation between two functions.

3. *Order* operates on functionally defined constituents (of the same unit).

4. *Conflate* creates an identity relation between two constituent functions; the result is a constituent function bundle.

5. *Preselect* does not create constituency itself, since it specifies a constraint on how to realize a function, but it presupposes the re-expression of a constituent function of one unit (e.g., the Agent of a clause) as the structure of another unit (e.g., Epithet ^ Thing of a nominal group). In other words, preselection is in terms of the overall constituency organization.

The functions themselves are (labels of) constituents. For example, (i) they are segmental in nature; no function can stretch over more than one segment unless it stands in a constituency relation to other functions (through the application of the Expand operator). (ii) They are clearly bounded segments of a categorical nature; there is no gradience (cline).

The use of the operators is summarized diagrammatically in Figure 12 to bring out their creation of, or reliance on, constituency.

Each constituent in an experiential layer of structure has a unique value in relation to the whole; the organization is multivariate. For example, a transitivity structure is organized into functions such as Senser Process Phenomenon Manner Locative. Since constituency is the experiential mode of structuring, experiential realization statements are usually well accommodated by multivariate constituency organization. For example, if a mental clause is metaphenomenal, there is a preselection stated in terms of one of the multivariate functions, the Phenomenon: "Phenomenon: noun-clause". If a clause is an unmarked intensive ascriptive one, the Process is constrained to be 'be' and the

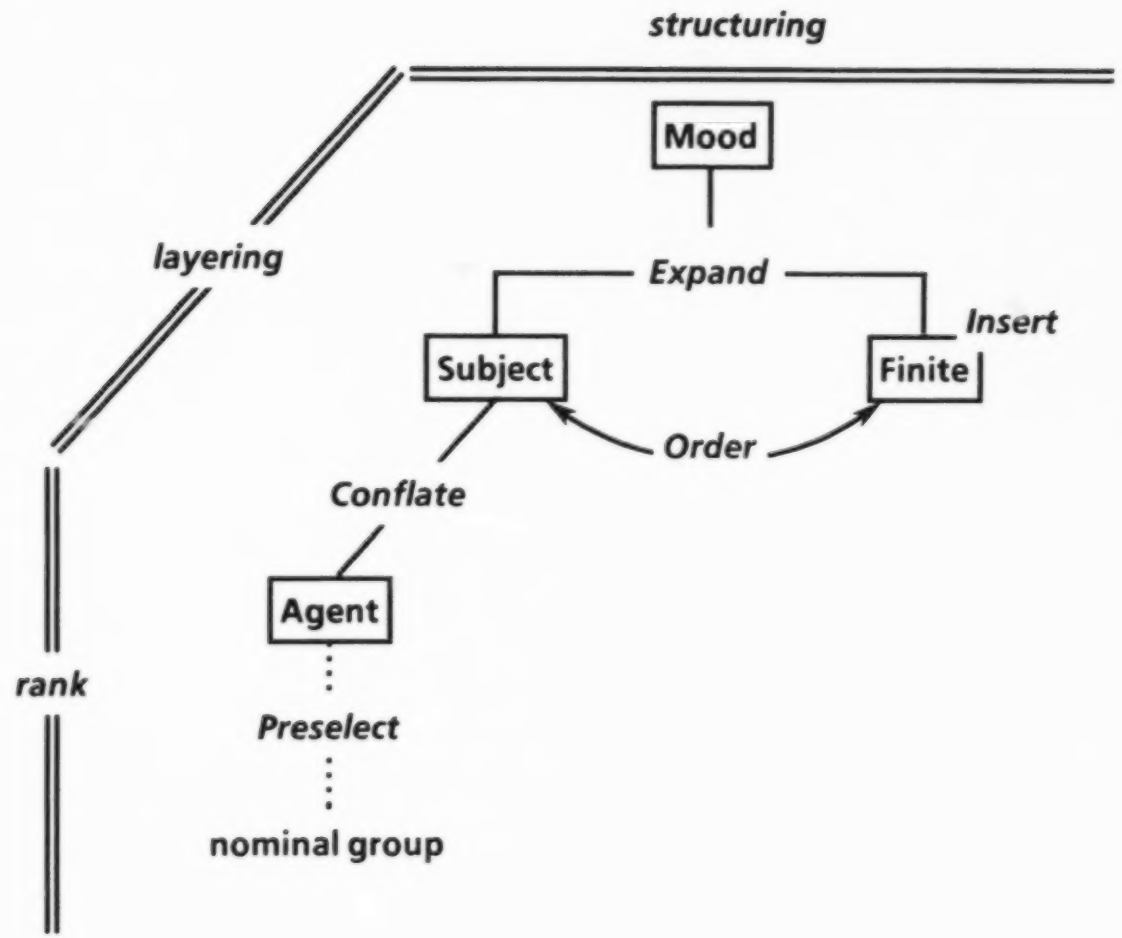


Figure 12: Realization operators and constituency

Attribute is a group with either a nominal or an adjectival head. Default orderings are also stated in terms of the multivariate constituent values. For example, we can state (Process Manner Locative), which means 'Process precedes Manner whic' precedes Locative'.

In contrast, realization statements other than those deriving from the experiential

metafunction are not as easily accomodated by multivariate constituency. For example, logical realizations are stated more naturally in terms of (univariate) interdependency than in terms of constituency: see Section 4.4 below.

The observation that the experiential mode of organization is (multivariate) constituency does not mean that there are no problematic cases. Linguists have sometimes felt a need to interpret structures as fusions/mergers and to leave constituent status indeterminate; such interpretations are outside the scope of Nigel's present set of realization operators.

4.2 Prosody

Agreement is an example of a mode of organization different from constituency; I would suggest it is what Halliday calls prosodic. It is the organization of grammatical structure, not as constituent segments, but as prosodies. If it is stated in terms of constituency, it typically has to be characterized as running across more than one constituent just as e.g., nasality may run across several phonological segments. Agreement is a case in point. It is, as already suggested, a mood characteristic rather than a Subject property or a Finite property per se, since it runs across these two constituents.

Prosodic organization includes:

mood number & person (agreement)

moodtag

polarity

reflexivization

key

Moodtag. Mood number & person and moodtag are closely related. If a moodtag is present, as in *You are coming out tonight, aren't you?*, the person and number selections run through both the mood and the moodtag. This prosody is represented diagrammatically in Figure 13.

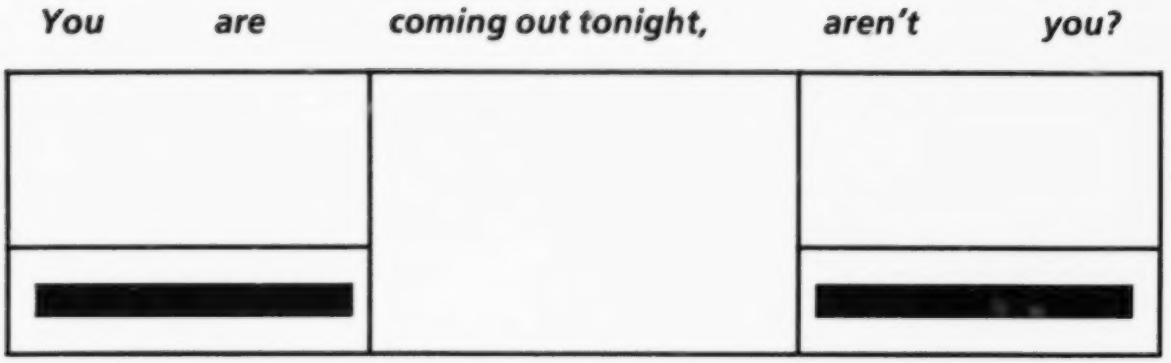


Figure 13: Mood and moodtag - number & person prosody

The existing realization operators cannot achieve more formally what is represented diagrammatically in Figure 13. Features can only be manipulated as segments associated with constituent functions; they cannot be introduced as prosodies.

Polarity. Polarity is similar: it is interpersonal in that it represents the speaker's denial or affirmation, agreement or disagreement, command or prohibition, etc. with respect to a proposition/proposal (cf. also Givon (1979: chapter on Negation)). Structurally, it is often not locatable in any one particular place. In "non-standard" English it is often realized wherever possible; the particular transitivity value does not matter:

I ain't never had no trouble with none of 'em

(from Labov (1970)

"Standard" English is of course quite similar in this respect. The difference is that the pattern is not *not ... never no none* as in the example above but rather *not ... ever*

any any with so-called non-assertive polarity items following *not*. The prosodic character remains; the "non-assertive" forms are typically used wherever possible. Diagrammatically, the polarity prosody can be represented in the same way as the mood person & number prosody; see Figure 14.

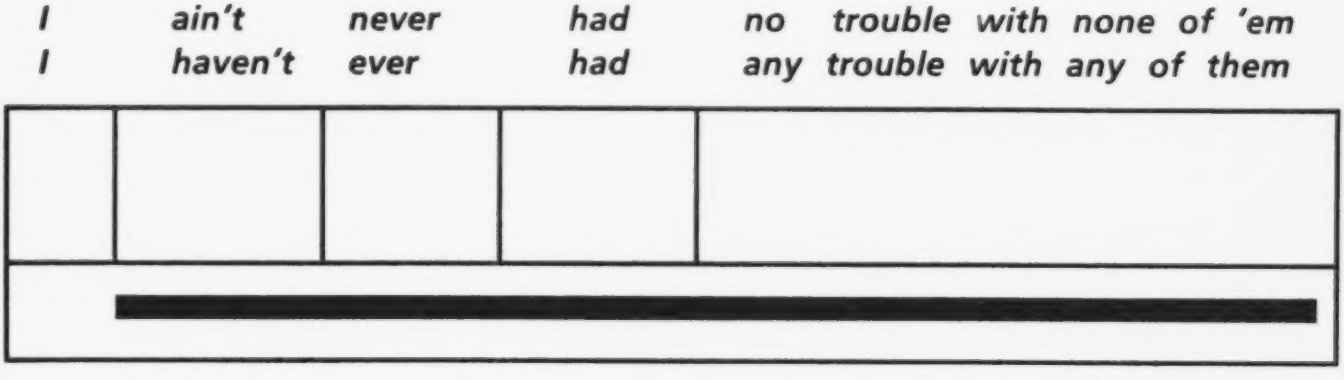


Figure 14: Polarity prosody

Just as before, there is no formal representation of the prosody, no realization operator that can be used. And just as before, this is to be expected since the realization operators are constituency-oriented rather than prosody-oriented.

Reflexivization. Reflexivization also seems to work as a prosody. Whenever appropriate, a constituent following the Subject of a clause is marked for identity of reference:

> Henry talked about himself to himself often in those days of intense loneliness

The particular value in the transitivity structure of the constituent reflexivized (Goal, Actor, Recipient, etc.) does not matter; what matters is whether the reflexive prosody

runs through it or not, and whether it can expound the reflexive prosody or not. Again, the formal problems in a treatment of reflexivization can be predicted from its prosodic nature. There is no easy way to use the Preselect operator to achieve reflexivization wherever appropriate.

Key. Key is perhaps more obviously realized prosodically than the other regions since it is realized by tone; cf. for example Halliday (1979).

Representation of prosody by constituency: Expand. The representational problems with prosodic organization are brought out when we try to capture the prosodies in constituency terms. Halliday has introduced a constituency interpretation of the clause as an interpersonal unit (the clause as exchange/interaction). This is the Mood ^ Residue structuring. Mood consists of Subject and Finite and Residue is, in principle, the rest (Predicator, Complements, and Adjuncts). These constituents can be used in stating the domains of the interpersonal prosodies, the domains of person & number, polarity, etc. This possibility is illustrated in the two diagrams of Figure Figure 15, first for the example from Figure 14 and then for an example also involving reflexivization.

The features 'speaker', 'reflexive', and 'non-assertive' have been associated with the Mood and Residue functions, not with the functions that are directly realized by groups (Adjunct, Complement, etc.), to indicate their domains in constituency terms. (I use the feature 'non-assertive' in the way it is used by Quirk et al. I intend it to be taken only as a short-hand approximation.)

This approach could be a constituency solution to the problem of representing grammatical prosodies. However, there is no formal correlate of the diagram cast in terms of realization operators. Presently, the Preselect operator can only be used to associate a feature with a grammatical function that is directly realized by the grammar. Both Mood and Residue are further expanded rather than directly realized. Furthermore, the diagram presupposes that a knowledgeable procedure of feature distribution will be activated where appropriate. For example, the feature 'non-

(a) *I haven't ever had any trouble with any of them*

Subj.	Finite: negative	Adjunct	Pred.	Complement
Mood		Residue: non-assertive		

(b) *I haven't ever asked myself any hard questions*

Subj.	Finite: negative	Adjunct	Pred.	Complement	Complement
Mood: speaker		Residue: reflexive, non-assertive			

Figure 15: Prosodic organization and constituency

assertive' should only be applied to adverbial and nominal groups that realize functions expanding Residue. The feature 'reflexive' would not behave like other features. It really means something like "check to see whether the nominal group is coreferential with the Subject; if it is, choose 'reflexive' in that group". And just as with the feature 'non-assertive', there is the issue of appropriate applicability.

4.3 Pulse

Halliday (1985) argues that the discontinuity in *They called the meeting off* arises for textual reasons. In information structure, the function New is assigned to the process of the clause in an unmarked way, which means that it should be the last open-class lexical item of the clause. If there is a complement, this can be achieved by "splitting" a phrasal verb, thereby creating discontinuity. The assignment of New is one of two kinds of textual prominence that affect the clause. It is a culminative kind of prominence (cf. Halliday (1979; 1985)), while the other is thematic prominence. Both may lead to representational problems:

Culminative prominence:	split Subject
	split Complement
	split Process
Thematic prominence:	thematic Minor complement (in prep. phrase)
	thematic participant from projected clause
	thematic Qualifier of symbolic Thing

Culminative prominence. The representational problem with the first type, culminative prominence, is essentially associated with discontinuity: a mood or transitivity constituent is split into two parts, one of which appears later than the other, presumably as unmarked New. (Interestingly, the part that is sequentially "delayed" in this way can usually be interpreted as a β in a logical structure.) The problem was illustrated above with a split Process. Here are some examples of the other types:

Anyone is a fool who tries to pat an alligator

I'll give anyone ten dollars who can help me find my cat

Questions remained about less fortunate enemies of the Marcos regime *(Time)*

The attempt was made to find a practical way to solve the problem without going into its theoretical depths (Tillich)

I call a person bad who lies and cheats and is unkind (Maugham)

(Some of these examples may involve grammatical metaphor. For example, the last example can perhaps be related to *I call a person bad || if he lies and cheats and is unkind.*)

Thematic prominence. The second type, thematic prominence, may lead to problems quite similar to those of the first type: an element is a constituent of one unit under one interpretation (usually transitivity or modification structure) and a constituent of another unit under a textual interpretation, more specifically as Theme. For example, *this house* is the Minirange of the prepositional phrase realizing the circumstance Locative (a transitivity function in the clause) and it is the Theme of the clause in which it appears initially:

This house we lived *in* for ten years

The constituency problem is diagrammed in Figure 16 (the@sign indicates the non-thematic position of the Minirange).

Notice that the constituency problem of the example is not an experiential one. Rather, the conflict arises as the two metafunctions, the textual and the experiential, are mapped onto one another through Theme selection. Textually, *this house* is a constituent of the clause, but experientially, it is a constituent of the prepositional phrase. (This kind of structure is very common in wh-clauses, of course, where the wh-item is the unmarked Theme: *Which house did you live in?*)

Why does the textual metafunction not "respect" the constituency organization created by the experiential metafunction? There are at least two related answers. One answer is that we simply have two different constituent structurings, the textual (Theme-Rheme) and the experiential (transitivity and minor transitivity); and that although they are normally *in phase* with one another, there are situations where the

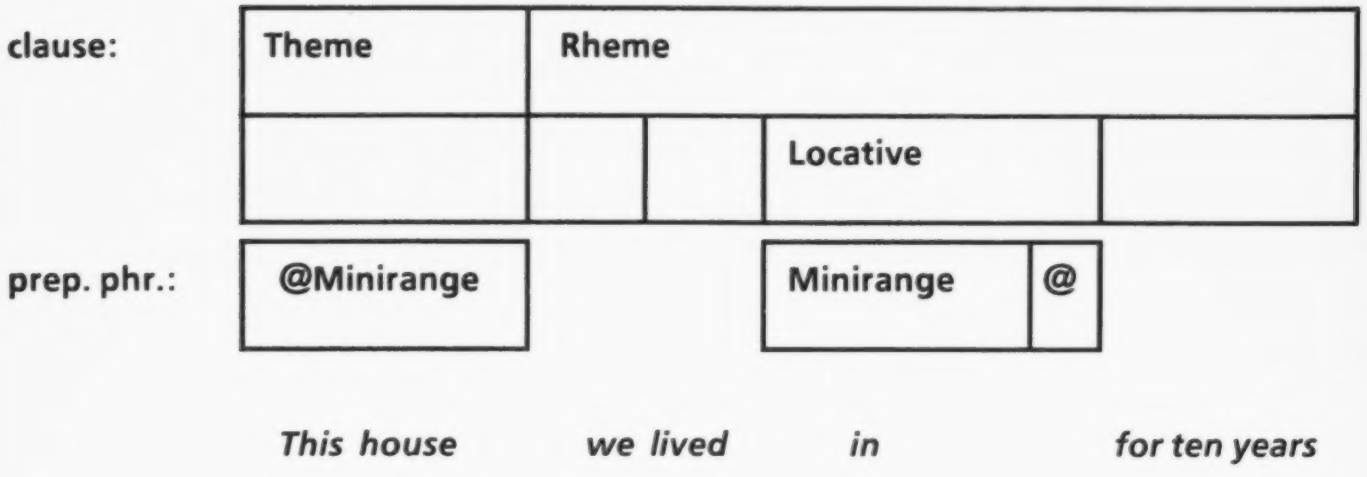


Figure 16: Theme and marked constituency

two structuring principles are out of phase with one another. This answer goes some way towards describing the situation in a principled way and it certainly seems better than merely observing that a constituent has been dislocated.

The second answer goes beyond distinguishing two distinct constituency principles. It says that the textual principle of structuring is not really a constituency one, but has a different character. I will call this kind of structure a pulse, because textual prominence can be thought of as a pulse that ripples through the experiential structure. Halliday calls this kind of structure periodic. The notion can be compared to Pike's wave.

There are at least three important reasons for the pulse or wave interpretation:

1. The pulse (wave) is periodic, with phases of prominence and phases of non-prominence. (There are in fact two pulses: the thematic pulse and the information pulse.)

2. The pulse (wave) is characterized by gradual transitions rather than sharp constituency boundaries.

3. The pulse (wave) is dynamic, running across the constituency hierarchy.

The third point in particular explains why textual structuring does not respect the hierarchic organization created by the experiential metafunction. Note that the theme pulse characterizes only clauses, not groups and phrases. Or rather, the pulse is built into the nominal and verbal group structures realized sequentially with a deictic-finite starting point (see Halliday (1985)), but there is no choice. The textual metafunction does not generate a constituency hierarchy of themes in the way the experiential metafunction generates a constituency hierarchy of phenomena decomposed into processes, participants, and circumstances, and then e.g., participants in their turn decomposed into things, epithets, and so on.

To represent textual structure graphically, Halliday uses a diminuendo-crescendo diagram. Consider the following example with multiple themes:

> In two or more years, both in London to which he went for the last part of the season and to pay a round of country house visits in the early autumn, and in Paris, where he had settled down, he knew everyone whom a young American could know.
>
> (Maugham)

The multiple themes can be represented graphically as follows (skipping part of the example to accomodate it in the figure); see Figure 17.

Representation of pulse by constituency: Expand. The thematic pulse affects interpersonal and experiential functions and it can be represented as if it were a constituent function expanded by the functions it affects. We have to treat the non-discrete pulse as if it were a discrete constituent Theme, finding a thematic boundary somewhere (see Halliday (1985: chapter 3)). In addition, we have to give up on representing the dynamic nature of the pulse and its disregard for experiential constituency. The expand operator can be used, Theme (Place) and Theme (Time), giving the following structure; see Figure 18.

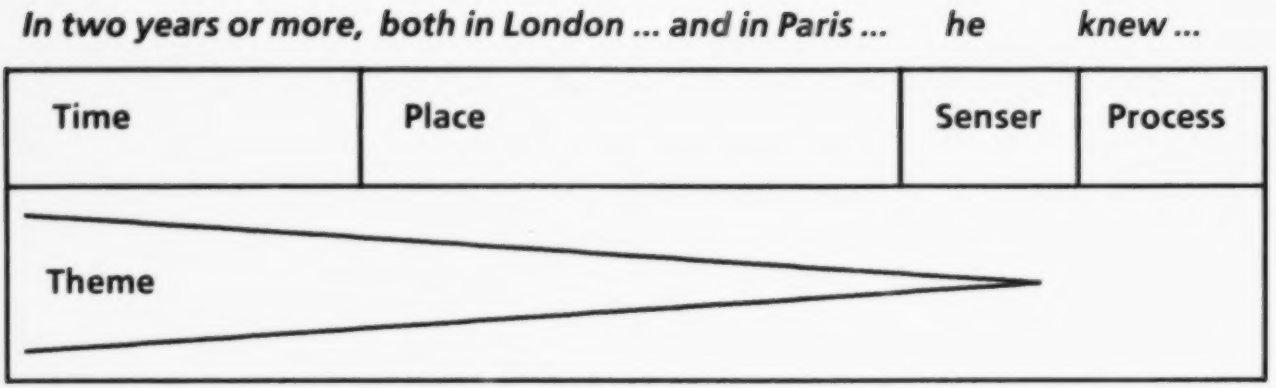


Figure 17: Thematic diminuendo part of the textual pulse -- graphic representation

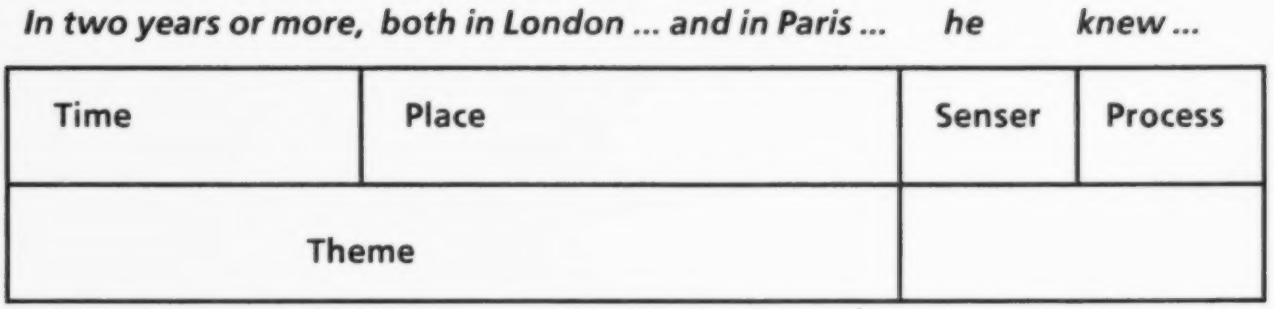


Figure 18: Representation of thematic pulse by constituency

4.4 Interdependency (chain)

The logical metafunction generates interdependency or chain structures; for a general discussion, see Martin (1986). When a logical element of structure is inserted, it is inserted as interdependent on the previous logical element and not as a constituent of some whole. The structure is thus like a chain. Each new link is defined in relation to the previous link. In principle, this is the structuring that characterizes complexes of units rather than simplex units: *Tom, Dick, and Harry* as opposed to *Tom; I went to Joe's Place and had a steak* as opposed to *I went to Joe's Place.* In particular, clause complexes and prepositional phrase complexes have interdependency structures, but clauses and prepositional phrases do not. Groups can, however, be interpreted as having interdependency structures (see Halliday (1985: ch. 6)). The structure of the nominal group complex *Tom, Dick, and Harry* can be diagrammed as follows in Figure 19.

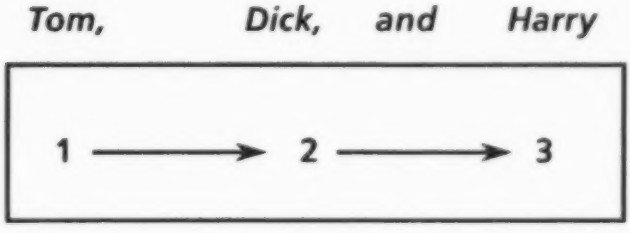


Figure 19: Graphic representation of interdependency structure

The functions in the interdependency structure do not have unique constituency values. Instead, the functions are values in a logical ordering such as 1 2 3 4; the value of each function is determined by its logical ordering. *Tom* is 1 because it is the first coordinate, *Dick* is 2 because it is the second coordinate, and *Harry* is 3 because it is the third coordinate.

The general representational problem with interdependency structures is that there are

currently no realization operators that can be used to specify them. In Figure 1, I used x -> y to mean 'insert y as interdependent on x', but there is no formal interpretation of it. The Insert operator discussed in Section 3.3.1 cannot be used to insert a logical function, since it inserts a function as a constituent of some unit and not as interdependent on another function. Examples of interdependency structures include

Clause complexes: clause coordination, etc.

Nominal group complexes, etc.

Modification in nominal groups

Tense in verbal groups.

As already mentioned in Section 4.1, realization statements deriving from the logical metafunction are easier to state in terms of interdependency than in terms of constituency. Experiential realization statements make reference to (multivariate) functions with unique values in relation to the whole. Logical realization statements make reference to (univariate) functions whose values reflect their logical ordering in interdependency. I will discuss clause tense and coordination only.

Tense. As an example of tense structure, I will take the verbal group *'ll have been going to be being tested* from Halliday's example below (Halliday (1976: 145)):

Can I use that machine when I come in at this time tomorrow?

No -- it's going to be being tested.

It'll have been going to be being tested every day for a fortnight soon.

The logical structure given to the verbal group includes tense and voice auxiliaries as follows in Figure 20. (The structure is a dependency structure and the letters of the Greek alphabet are used to represent the logical functions; the structure is Figure 19 is a paratactic one. The difference is discussed in Halliday (1985: Chapter 7).)

Realization statements have to specify the sequence of the functions of the verbal

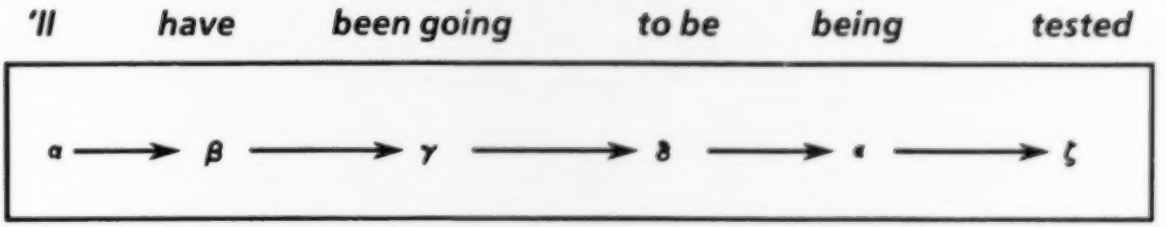


Figure 20: Logical interdependency structure of verbal group

group and they have to specify the inflectional forms. Both specifications are stated most naturally in terms of the dependency ordering. (i) A dependent function follows the function it is dependent on: β follows alpha, and so on. (ii) The inflectional form of the verb realizing a dependent function is determined by the realization of the function it is dependent on: *have* (β) determines the past participial form of *been going to* (γ). (For a thorough discussion of dependency and auxiliary order, see Schachter (1981).)

Clause coordination. Coordination is similar to tense in respect to what realization statements need to refer to. The sequence of coordinated clauses reflects the order in which they are coordinated. Furthermore, structural ellipsis is determined by interdependency relations. For example, Subject ellipsis works forwards in the interdependency structure (progressive ellipsis) while Complement ellipsis works backwards (regressive ellipsis). Both kinds of ellipsis make reference to direction in the interdependency structure, not to constituency. Furthermore, progressive ellipsis of Subject, for example, can continue for as long as the interdependency chain continues.

It is also worth noting that structural markers such as *and* and *or* are specified by reference to the ordering of the chain, before the last link. In this respect, they contrast with both case marking and marking by preposition.

5 MODES OF MEANING AND MODES OF CHOICE

The main point of the previous section is that different kinds of meaning (experiential, interpersonal, textual, and logical) are associated with different kinds of structure (constituence, prosody, pulse, and interdependence). This enables us to characterize the representational problems pertaining to realization operators in a principled way: the operators are constituency-oriented, and other kinds of structure will cause various types of problems for them. Although we can represent prosodies, pulses, and interdependencies diagrammatically, there are at present no realization operators that can introduce and operate on prosodies, pulses, and interdependencies. For example, insertion means insertion as a constituent in the structure of a unit; there is no insertion operator for interdependents (no 'insert x as interdependent on y').

The presentation so far has focused on problems in structural representation. However, I think that the different modes of meaning are reflected not only in the syntagmatic organization (defined by the realization operators) but also in the paradigmatic organization, i.e., in the network of systems.

The experiential mode of meaning leads to multivariate constituency structures. The systemic organization is similar: systems have unique multivariate feature values and are ordered hierarchically in delicacy. The other metafunctions raise representational issues. This is true particularly of logical and textual systems. I will look at them in that order. (I will not discuss interpersonal systems and the representation of interpersonal clines and prosodic features.)

5.1 Logical: recursive systems

Logical systems have long been recognized as having special status. The representational problem that arises from this special status was mentioned in Section 2 and I will return to it now. Logical systems are assumed to be (linearly) recursive. The recursion is represented by a loop back to the entry condition in the system, as in Figure 1.

Problem, systemic recursion. Henrici's (1966) diagnosis of the problem with

recursive systems has already been quoted (Section 2). They cannot be represented formally in the same way as "conventional systems". The output of the system also provides an input to it (the loop in Figure 1). Systemic features are distinguished by their names, and since the feature complex in the system in Figure 1 can be chosen more than once, there is a naming problem. It is not clear how the feature chosen the first time is distinct from the feature when it is chosen the second time, and so on. This is related to Henrici's remarks about partial ordering. Moreover, how is a realization statement to be applied recursively when a system is entered more than once?

Towards a solution. One approach towards a solution is in terms of the type-token distinction. We can think of a system as a system type. Each time it is entered a system is instantiated as a token of the type. For conventional systems, there would only be one token per pass through the grammar, but recursive systems would allow multiple tokens. These tokens would have to be ordered by a counter so that each token has a unique place in the ordering: token 1, token 2, token 3, and so on. (Note that this approach appeals to the generation algorithm for traversing the system network. If we see the logical resources as essentially dynamic rather than static, the appeal to the generation algorithm follows naturally. For the distinction between dynamic and static systems, cf. Martin (1985).)

In this approach, the difference between multivariate and univariate is the difference between instantiations of multiple types and multiple instantiations of the same type. For example, in the multivariate structure Actor Process Goal Recipient *(I gave the report to Henry)*, the participants are inserted by realization statements associated with three different systems. In the univariate structure 1 2 3 [and] 4 *(Tom, Dick, Harry, and Henry)*, the coordinates are inserted by the same system in successive instantiations of it.

5.2 Textual: variable/dynamic systems

Textual structure can be characterized as a pulse or a wave. Is there a systemic correlate, a system pulse comparable to the structure pulse? Arguably there is, if the pulse is interpreted as dynamic/variable in nature: a possible correlate is a dynamic/variable system.

As an example, consider theme selection. The issue here is how to represent the resources of thematization. What are the thematization options? For example, what are the experiential theme options from the transitivity of a clause? The answer depends on transitivity choices (e.g., is the clause middle or effective, benefactive or non-benefactive, ranged or non-ranged?); on related circumstantial choices (e.g., is there a Locative, a Cause, a Manner, etc.?); and also on voice choices (e.g., is the Agent or the Medium the Subject and thus the unmarked Theme candidate?).

The theme potential varies with the transitivity choices. It can be characterized, at least informally, as a (set of) dynamic/variable system(s). For example, in a middle (non-ranged) clause, there is only one participant Theme candidate, the Medium; in an effective (agentive) clause, there are two, the Medium and the Agent; and in a benefactive clause, there are three, the Medium, the Agent, and the Beneficiary. Similarly, there are as many circumstantial Theme candidates in a clause as there are circumstances.

The notion of a dynamic or variable system is merely a way of thinking about a problem like theme selection; it is not a solution. It raises the issue of process-oriented accounts mentioned in Section 1. It does not seem unreasonable that some systems are inherently dynamic rather than static, particularly systems deriving from the textual metafunction, i.e., the enabling metafunction concerned with the process of meaning itself.

6 CONCLUSION: LOOKING FOR SOLUTIONS

I have discussed various problems of representation and mentioned a few possible solutions. I have tried to diagnose the problems in terms of the four different metafunctional modes of organization, suggesting specific problems associated with each of the non-constituency modes.

There are, of course, problems in lexicogrammatical representation not discussed in this paper. They include: theme predication, ellipsis; grammatical metaphor, play on the system; and the accomodation of certain lexical information (e.g., collocation, and lexical functions), lexical items not corresponding to grammatical constituents, lexical items at ranks above word rank, and so on. These problems are not caused simply by non-constituency modes of organization, as are the problems I have focused on in the paper, but they can also be explained in principled ways. For example, one significant set of problems is caused by re-representation within the linguistic system itself. The set includes textual organization re-represented as identifying relational transitivity organization (theme predication and theme nominalization), as well as grammatical metaphor in general, where e.g., one transitivity structure is re-represented by another transitivity structure (see Halliday (1985: Chapter 10)).

Where can we find solutions? If we take natural language as a model for our metalanguage, our "language" of representation, two of the methods for expanding the potential that suggest themselves are metaphor and borrowing.

(i) Metaphor. The prevailing metaphor is in a sense the source of the problems discussed in this paper. It is the constituency metaphor: treat all modes of structuring as if they were constituency. For example, as I showed in Section 4.2 (Representation of prosody by consituency), a prosody running across constituents can be represented as if it were a function expanded by these constituents and a pulse can also be represented as if it were a function expanded by the functions affected by the pulse (Section 4.3: Representation of pulse by constituency).

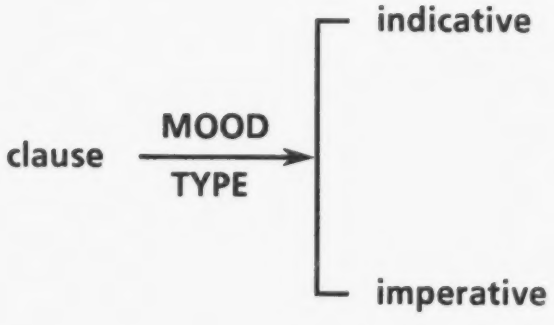
(ii) Borrowing. We may borrow from other systemic "dialects": problems for Nigel

may not be problems for Fawcett's and Hudson's grammars, for example. We may also borrow resources from other grammatical frameworks to represent aspects of systemic theory. Some frameworks are typologically too far away from the systemic one; we are not likely to find much of any use to us in transformational grammar. Other frameworks are typologically much closer and in some cases there is a long tradition of exchange, e.g., stratificational linguistics. One kind of representation close enough to be of considerable interest is Kay's Functional Unification Grammar. I have not explored it in this paper, but have pointed to places where a FUG treatment is developed in Kasper's paper in this volume. Finally, we may borrow representational resources from other disciplines which have strong notions of e.g., processes and non-constituency organizations.

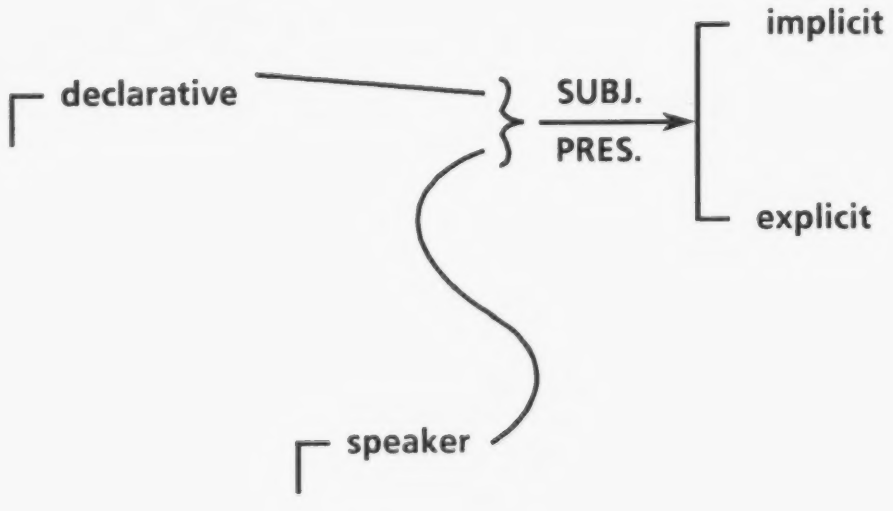
APPENDIX: REPRESENTATIONAL CONVENTIONS

[i] System network

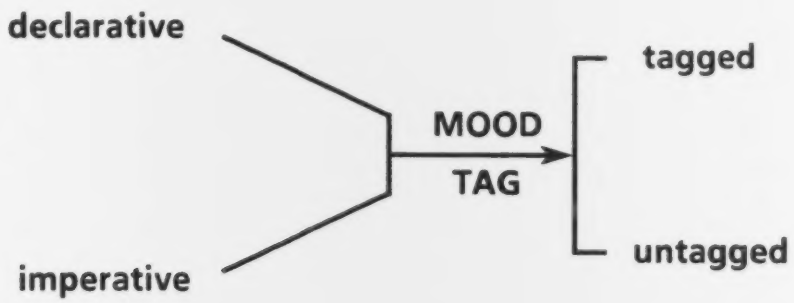
[a] The MOOD type system has the entry condition (input features) clause and the terms (output features) indicative vs. imperative. If clause has been chosen, either indicative or imperative is chosen.



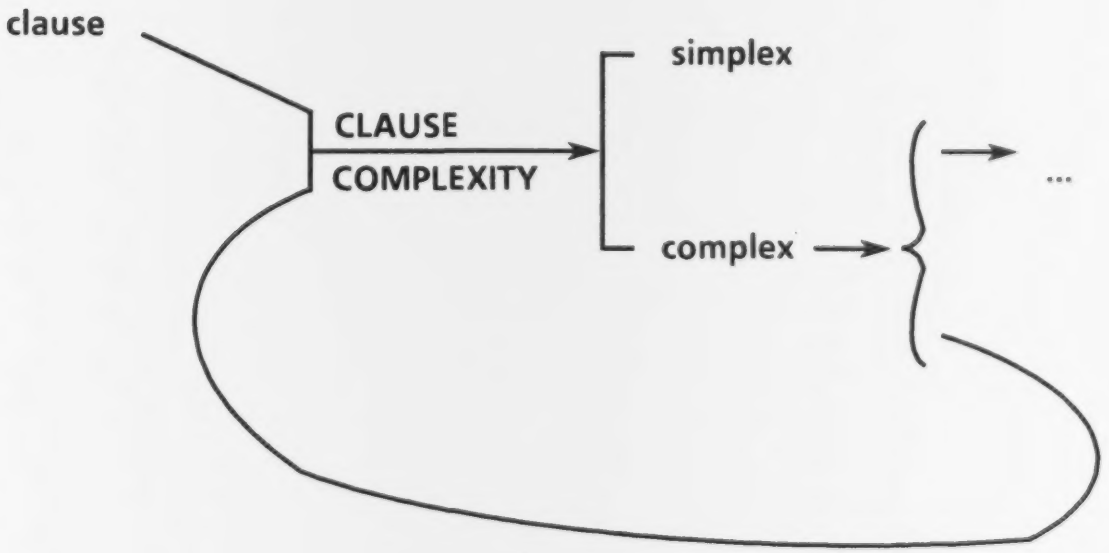
[b] The [DECLARATIVE] SUBJECT PRESUMPTION system has the entry condition (the input features) declarative and speaker and the terms (output features) implicit vs. explicit. If the features declarative and speaker have been chosen, either implicit or explicit is chosen.



[c] The MOOD TAG system has the entry condition (input features) declarative or imperative and the terms (output features) tagged vs. untagged. If the feature declarative or imperative has been chosen, either tagged or untagged is chosen.



[d] The CLAUSE COMPLEXITY system has the entry condition (input features) clause or complex and the terms (output features) simplex vs. complex. If the feature complex is chosen, the entry condition of the system has been satisfied again. This is a so-called recursive system.



[ii] Realization statements

Name of operator		Description
(Insert Subject)	+Subject	Function inserted as constituent of the structure of the unit being specified
(Withhold Subject)	*Subject	Function that is present by default is withheld from the function structure [Discussed, but not implemented in Nigel]
(Depend 1 2)	1 -> 2	Function 2 inserted as interdependent on function 1 [Used as an illustration, but not implemented in Nigel]
(Order Subject Finite)	Subject^Finite	One function ordered to precede another function
(Include Goal Process)	Process <Goal>	One function is included within another function [Sometimes used in systemics, but not implemented in Nigel]
(Expand Mood Finite)	Mood (Finite)	One function is expanded to have another function as constituent
(Conflate Subject Agent)	Subject/Agent	One function is conflated with another function to form part of the same constituent function bundle
(Preselect Subject singular)	Subject: singular	A function is preselected for a feature; the realization of the function is constrained to display that feature

BIBLIOGRAPHY

1. Allerton, D. J., Carney, E., and Holdcroft, D. (eds.). 1979. *Function and Context in Linguistic Analysis: Essays Offered to William Haas.* Cambridge University Press.

2. Appelt, D. 1983. TELEGRAM: A grammar formalism for language planning. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics.*

3. Bateman, J. 1985. A Systemic Grammar of Japanese. MS, Kyoto University.

4. Benson, J. D., and Greaves, W. S. (eds.). 1985. *Systemic Perspectives on Discourse, Volume 1.* Ablex Publishing Corporation.

5. Benson, J. D., and Greaves, W. S. (eds.). To appear. *Systemic Functional Approaches to Discourse.* Ablex Publishing Corporation.

6. Davey, A. 1978. *Discourse Production.* Edinburgh University Press.

7. Fawcett, R. P. 1980. *Cognitive Linguistics and Social Interaction.* Julius Groos Verlag Heidelberg and Exeter University.

8. Fawcett, R. P. 1984. System networks, codes, and knowledge of the universe. In Fawcett, Halliday, Lamb, and Makkai (eds.).

9. Fawcett, R. P., Halliday, M. A. K., Lamb, S. M., and Makkai, A. 1984. *The Semiotics of Culture and Language.* Volume 2: *Language and other Semiotic Systems of Culture.* Frances Pinter (Publishers).

10. Fawcett, R. P. 1986. What makes a 'good' system network good? -- Four pairs of concepts for such evaluations. To appear in Benson and Greaves (eds.).

11. Givon, T. 1979. *Understanding Grammar.* Academic Press.

12. Halliday, M. A. K. 1961. Categories of the theory of grammar. *Word*, 17:3.

13. Halliday, M. A. K. 1965. Types of structure. In Halliday and Martin (eds.).

14. Halliday, M. A. K. 1966. The English verbal group. Reprinted in Halliday (1976).

15. Halliday, M. A. K. 1976. *Halliday: System and Function in Language.* Selected papers edited by G. Kress. Oxford University Press.

16. Halliday, M. A. K. 1979. Modes of meaning and modes of expression: Types of grammatical structure, and their determination by different semantic functions. In Allerton et al. (eds.).

17. Halliday, M. A. K. 1985. *A Short Introduction to Functional Grammar.* Edward and Arnold.

18. Halliday, M. A. K., and Hasan, R. 1985. Language, context, and text: Aspects of language in a social-semiotic perspective. Deakin University.

19. Halliday, M. A. K., and Martin, J. R. 1983. *Readings in Systemic Linguistics.* Batsford.

20. Henrici, A. 1966. Some notes on the systemic generation of a paradigm of the English clause. In Halliday and Martin (eds.).

21. Hudson, R. A. 1971. *English Complex Sentences: An Introduction to Systemic Grammar.* North Holland.

22. Kay, M. 1979. Functional grammar. In *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society.*

23. Labov, W. 1970. Negative attraction and negative concord. In *Labov, Language in the Inner City: Studies in the Black English Vernacular.* The University of Pennsylvania Press, Inc.

24. McCord, M. C. 1975. On the form of a systemic grammar. *Journal of Linguistics,* 11.

25. McKeown, K. 1982. The TEXT system for natural language generation: An overview. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics.*

26. McKeown, K. 1985. *Text Generation. Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press.

27. Mann, W. C. 1983. A linguistic overview of the Nigel text generation grammar. In *The Tenth LACUS Forum.*

28. Mann, W. C., and Matthiessen, C. 1984. The Nigel realization operators. *Network* 7.

29. Mann, W. C., and Matthiessen, C. 1985. A demonstration of the Nigel text generation computer program. In Benson and Greaves (eds.).

30. Martin, J. R. 1985. Process and text: Two aspects of human semiosis. In Benson and Greaves (eds.).

31. Martin, J. R. 1986. Recursive systems in English: Towards a functional interpretation. MS, Department of Linguistics, Sydney University. To appear in Benson and Greaves (eds.).

32. Matthiessen, C. 1985. The systemic framework in text generation: Nigel. In Benson and Greaves (eds.).

33. Matthiessen, C. 1983. Systemic grammar in computation: The Nigel case. In *Proceedings of the First Conference of the European Chapter of the Association for Computational Linguistics.*

34. Parret, H. 1974. *Discussing Language.* Mouton.

35. Pike, K. L. 1959. Language as particle, wave, and field. *The Texas Quarterly,* 2:2.37-54.

36. Quirk, R., Greenbaum, S., Leech, G., and Svartrik, G. 1972. *A Grammar of Contemporary English.* Longman.

37. Schachter, P. 1981. *Explaining Auxiliary Order.* Indiana University Linguistics Club.

Systemic Grammar and Functional Unification Grammar

Robert Kasper
USC / Information Sciences Institute

1 Background

As this paper is being presented in the context of a workshop on systemic linguistics, it will assume some familiarity with systemic grammar. No familiarity with computer science or Functional Unification Grammar will be assumed. This section provides a historical perspective to what follows.

Functional Unification Grammar (FUG) was first formulated by Kay (1979) as a creative method for experimentation in computational linguistics. It is a framework that represents a grammar in a form that is simultaneously readable by linguists and suitable for use by computer programs that generate or analyze text. Unlike many previous grammatical formalisms, which required a different representation for analysis than for generation, FUG is intended to be neutral in this respect.[1] In earlier work, FUG was also called Functional Grammar or Unification Grammar.

The notation of FUG looks very different from that of Systemic Functional Grammar (SFG); but if we look beyond the differences of format, we find that FUG and SFG make many of the same assumptions about language and grammar. The similarities are due, in part, to the fact that when Martin Kay (1979) formulated FUG he was responding to many of Michael Halliday's ideas.

Both FUG and SFG organize a grammatical description around feature choices, stressing the paradigmatic aspects of language structure. Structural realizations are stated as consequences of these feature choices. Both represent the function of constituents explicitly in addition to the position of constituents in a sequence. Both provide a means for multifunctional description, making possible the expression of textual and interpersonal roles in addition to ideational roles.

Given these similarities, I was led to consider FUG as a promising tool for constructing an analysis program based on systemic grammar.[2] A more careful investigation has shown that it is possible to represent most of a systemic grammar in FUG without any significant loss of clarity. Some natural extensions to the FUG framework handle the residue (see Section 4.3). Such rerepresentation of a grammar is motivated by Halliday's idea (1964) that the form of notation used to express a grammar should suit the task for which it is to be used. FUG notation is especially well suited to the task of computational processing. If we are able to represent SFG in FUG, then we have at our disposal the computational tools that have already been developed for FUG.

[1] Any modifications to the generation grammar that are needed for the computational analysis program should be performed automatically by the program, as explained in (Kay, 1985).

[2] Bill Mann originally suggested that FUG might be an appropriate tool for this purpose.

The grammar being used for the study is Nigel (Mann & Matthiessen, 1985), a large grammar of English that is the result of a collective effort with significant contributions by several systemic linguists. This grammar has been used as part of a computer program for text generation, but there has previously been no program for text analysis.

Section 2 provides an introduction to the basic concepts of FUG. Section 3 demonstrates how the systems of the Nigel grammar can be represented in FUG notation. Section 4 explains some of the differences between FUG and SFG. Section 5 shows how the FUG framework provides some potential solutions to problems in systemic notation. Section 6 describes a computational analysis program, which uses the FUG representation of the Nigel grammar, and indicates some of its uses.

2 The Ingredients of FUG

This section is a brief introduction to FUG. For a more thorough treatment the reader is referred to Kay's work (Kay 1979, 1985).

2.1 Functional Descriptions

In FUG each linguistic object is represented by a *functional description* (hereafter abbreviated as FD). FDs contain lists of attributes, each composed of a label/value pair. Figure 1 shows a simple FD, which contains three attributes, with the labels Rank, PrimaryClass, and Animacy. An equal sign (=) is used to separate the attribute label, which appears on the left, from the value, which appears on the right. The order of attributes in a description is not usually significant.

FDs are also called *feature structures* in the computational linguistics literature. In this paper the features of an FD will be called attributes, in order to distinguish them from the features of systemic grammar.

FDs can be used to represent linguistic objects of any size, such as words, phrases, clauses, and even grammars. Each value in Figure 1 is a simple atomic symbol. An *atom* is a string of characters defining a value that is not further decomposed. Note that the case of the letters used in atoms can be significant. Thus, *AGENT* and *agent* would be regarded as different symbols. The values of attributes may also be FDs, allowing one FD to be embedded inside others. Figure 2 shows how embedded descriptions may be used to describe groups that are constituents of a clause. The FD of Figure 1 is used as the value of the AGENT attribute in the clause description. Since an FD may contain values that are themselves instantiated by FDs, a functional description can be viewed as a hierarchy of label/value pairs.

$$\begin{bmatrix} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Nominal} \\ \text{Animacy} = \text{Animate} \end{bmatrix}$$

Figure 1: A simple functional description

$$\left[\begin{array}{ll} \text{Rank} = \text{Clause} & \\ \text{AGENT} = & \left[\begin{array}{l} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Nominal} \\ \text{Animacy} = \text{Animate} \end{array}\right] \\ \text{PROCESS} = & \left[\begin{array}{l} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Verbal} \\ \text{Agency} = \text{Effective} \end{array}\right] \end{array}\right]$$

Figure 2: Embedded Functional Descriptions

2.2 Unification

Unification is an operation that combines two FDs, reconciling them into a single structure. The result of unification is an FD that contains all attributes of both of the original descriptions provided that they are compatible. The criteria of compatibility depend on the type of value. Simple atomic values are compatible only when they are identical. Values instantiated by complex FDs are compatible when all of their attributes are compatible. If a particular attribute is present in both FDs, then its values from both FDs are unified. Those attributes which are present only in one of the FDs are retained in the result. Thus, unification is a recursive process, terminating when all embedded attributes have been unified. If any of the attributes fail to unify, then the entire unification is a failure.

For example, Figure 5 is the FD that results from the unification of the FDs in Figures 3 and 4. The attributes having labels Rank and PrimaryClass appear in both FDs and they are compatible, because they have the same atomic value. The remaining attributes of each FD do not appear in the other, so they are simply copied into the result.

2.3 Paths

Paths are expressions that refer to the values of particular attributes in a FD. A path is composed of a sequence of labels enclosed in angle brackets. The value denoted by a path is found by a recursive procedure that extracts the value of its first attribute label, and applies the remainder of the path to that value.

Let us illustrate this procedure by considering the value of the path <AGENT Rank> when it is applied to the FD of Figure 2. First, we locate the value of the attribute of the FD that has the label, AGENT, identical to the first label of the path. The value of this attribute is itself a FD composed of three attributes. Next, we use the second label of the path, Rank, to locate one of these attributes, *Rank = Group*, and extract its value. Thus, the path refers to the value *Group*.

In general, each label of the path is used to locate attributes of successively embedded FDs. When the last label of the path has been used to locate an attribute, the value of that attribute is taken to be the value denoted by the path. If any of the labels of the path cannot be found in the FD, then the path has the special value *NIL*.

A path may be used within the description of a grammar as the value of another attribute in order to indicate the unification of two attributes. For example, in the FD of Figure 6, the path <AGENT> is used as the value of the SUBJECT attribute. This signifies that the value of the SUBJECT attribute is to be unified with value of the AGENT attribute.

$$\text{AGENT} = \begin{bmatrix} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Nominal} \\ \text{Animacy} = \text{Animate} \end{bmatrix}$$

Figure 3: FD for AGENT.

$$\text{SUBJECT} = \begin{bmatrix} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Nominal} \\ \text{Number} = \text{Singular} \\ \text{Person} = 3 \\ \text{Lex} = \text{John} \end{bmatrix}$$

Figure 4: FD for SUBJECT.

$$\text{AGENT} = \text{SUBJECT} = \begin{bmatrix} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Nominal} \\ \text{Animacy} = \text{Animate} \\ \text{Number} = \text{Singular} \\ \text{Person} = 3 \\ \text{Lex} = \text{John} \end{bmatrix}$$

Figure 5: Unification of AGENT and SUBJECT.

$$\begin{bmatrix} \text{Rank} = \text{Clause} \\ \text{AGENT} = \begin{bmatrix} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Nominal} \\ \text{Animacy} = \text{Animate} \end{bmatrix} \\ \text{SUBJECT} = \langle \text{AGENT} \rangle \end{bmatrix}$$

Figure 6: FD containing a path as the value of an attribute.

2.4 Patterns

A special type of attribute, which has the label *pattern*, describes how a function is realized by its constituents. Patterns specify the ordering of constituents with respect to each other. Pattern values contain a list of elements enclosed by parentheses. An element of a pattern may be the label of an attribute, which stands for a constituent function, or a special symbol (ellipsis) that indicates that any number of constituents may intervene in the pattern at that point.

For example, the pattern shown in Figure 7 specifies that SUBJECT and PROCESS are constituents of the clause represented by this FD, that SUBJECT is ordered immediately before PROCESS, and that any number of other constituents may come before SUBJECT or after PROCESS. More complicated patterns are possible, but the details will be omitted here.

2.5 Disjunctions

FDs may also contain disjunctions, which represent sets of alternative descriptions. The alternatives of a disjunction are surrounded by curly braces (i.e., {, }) in the most common notation, as shown by the value A in Figure 8. When the disjunctive value, A, is unified with another value, B (also shown in Figure 8), at least one of the alternatives of A must be compatible with B in order for the unification to be successful. If two or more alternatives are compatible with B, then the result of the unification is a disjunction containing the result of unifying B with each compatible alternative. Only one alternative of A is compatible with C. In this case, the result is a non-disjunctive value formed by unifying C with that alternative.

2.6 Representing a Grammar

We now have enough basic ingredients to state how a grammar can be represented by an FD. For each unit described by the grammar (such as a clause or group), the grammar will contain a FD giving its attributes. In general, this FD will be quite large, containing alternatives for all possible values of each attribute. In systemic terms, each attribute corresponds to the description of a feature or a function. An attribute that describes a feature usually has an atomic value. An attribute that describes a function usually has a complex FD as a value, giving the attributes of the constituent which realizes the function. By representing all possible alternative values for each attribute, an FD can be written to express the grammatical potential of a language, in a way analogous to a system network.

$$\left[\begin{array}{l} \text{Rank} = \text{Clause} \\ \text{pattern} = (\ldots \text{SUBJECT PROCESS} \ldots) \\ \text{SUBJECT} = \left[\begin{array}{l} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Nominal} \end{array}\right] \\ \text{PROCESS} = \left[\begin{array}{l} \text{Rank} = \text{Group} \\ \text{PrimaryClass} = \text{Verbal} \end{array}\right] \end{array}\right]$$

Figure 7: FD containing a pattern attribute.

$$A = \left\{ \begin{bmatrix} \text{number} = \text{singular} \\ \text{person} = 1 \end{bmatrix} \begin{bmatrix} \text{number} = \text{singular} \\ \text{person} = 2 \end{bmatrix} \begin{bmatrix} \text{number} = \text{plural} \end{bmatrix} \right\}$$

$$B = \begin{bmatrix} \text{number} = \text{singular} \\ \text{form} = \text{unmarked} \end{bmatrix}$$

$$A = B = \left\{ \begin{bmatrix} \text{number} = \text{singular} \\ \text{person} = 1 \\ \text{form} = \text{unmarked} \end{bmatrix} \begin{bmatrix} \text{number} = \text{singular} \\ \text{person} = 2 \\ \text{form} = \text{unmarked} \end{bmatrix} \right\}$$

$$C = \begin{bmatrix} \text{number} = \text{plural} \\ \text{form} = \text{unmarked} \end{bmatrix}$$

$$A = C = \begin{bmatrix} \text{number} = \text{plural} \\ \text{form} = \text{unmarked} \end{bmatrix}$$

Figure 8: Unification with Disjunctive Values

3 Representing Systemic Grammar in FUG

This section demonstrates by example how systemic grammar can be translated into the notation of FUG. Figure 9 shows the feature choices for two systems (MoodType and IndicativeType) from the clause network of the Nigel grammar. Figure 10 shows the translation of these systems into FUG notation.

3.1 Systems and Feature Choices

Each systemic choice is encoded by a disjunction that expresses the alternatives of the system. Each alternative of the disjunction contains an attribute with one of the systemic features that can be chosen in that system as its value. The system name is used as the label of that attribute. Notice that the symbol ':=' is used (in place of '=') to designate all attributes that correspond to systemic features. The reason for this notation is explained in Section 4.3.

The alternatives may also contain consequences of choosing a particular feature, including realizations and the representation of other systems that may be entered when that feature is chosen. Thus, delicacy can be expressed in FUG by embedding. The IndicativeType system is entered when the Indicative feature is chosen in the MoodType system, and it is represented by a FD embedded within the description of MoodType.

3.2 Realization

Figures 11 and 12 add the realization components of these two systems. This section explains the representation in FUG notation of five common systemic realization operators: insertion, preselection, classify, order, and conflation.

Insertion is indicated in FUG by a pattern containing the name of the inserted function. For example, when the Imperative feature is chosen, the NONFINITIVE function is inserted. This pattern states that the function must be present as a constituent of the clause, but it does not place any constraint on how NONFINITIVE may be ordered with respect to other constituents of the clause.

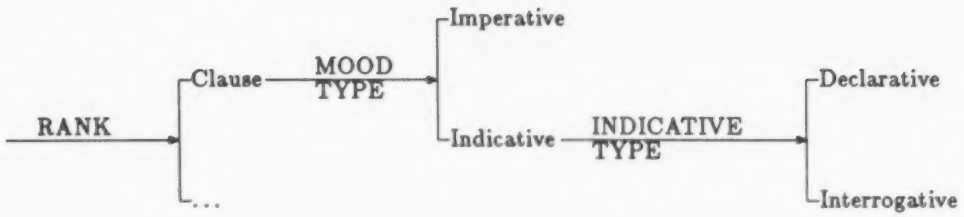


Figure 9: The MoodType and IndicativeType Systems

```
[ Rank = Clause
  ( { [ MoodType := Imperative ]
    | [ MoodType := Indicative
        ( { [ IndicativeType := Declarative ]
          | [ IndicativeType := Interrogative ] } ) ] } ) ]
```

Figure 10: The MoodType and IndicativeType Systems in FUG

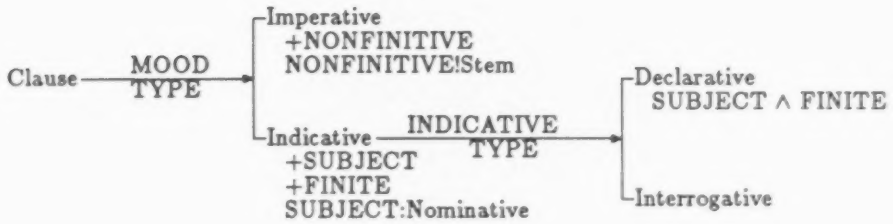


Figure 11: MoodType and IndicativeType Realizations

```
[ Rank = Clause
  ( { [ MoodType := Imperative
        pattern = (... NONFINITIVE ...)
        NONFINITIVE = [ Form = Stem ] ]
    | [ MoodType := Indicative
        pattern = (... SUBJECT ...)
        pattern = (... FINITE ...)
        SUBJECT = [ Case = Nominative ]
        ( { [ IndicativeType := Declarative
              pattern = (... SUBJECT FINITE ...) ]
          | [ IndicativeType := Interrogative ] } ) ] } ) ]
```

Figure 12: MoodType and IndicativeType Realizations in FUG

Preselection is translated into FUG by specifying an FD for the given function. This FD constrains one attribute of the function so that it must have the value of the preselected feature. For example, when Indicative is chosen, the SUBJECT of the clause is preselected to have the Nominative feature. In Figure 12, the attribute labeled by SUBJECT has a value that is an FD containing the statement *Case = Nominative*. The label, *Case*, is taken from the name of the system that selects the feature, *Nominative*, in the grammar for nominal groups.

Classify, the operator in Nigel that preselects lexical features, is translated by the same method as preselection, as shown by the classification of NONFINITIVE as a Stem. There is no formal distinction between lexical features and grammatical features in FUG, so preselection and classification are represented in exactly the same way. This is as things would be if Nigel used a fully systemic lexicon.

Order is translated into FUG by a pattern that contains the names of the functions in the required order. For example, when the Declarative feature is chosen, an ordering relation is specified requiring SUBJECT to immediately precede FINITE. The dots on either side of these functions are necessary, because this ordering relation does not specify any constraint on what may occur to the left of SUBJECT or to the right of FINITE.

Conflation is encoded precisely by unification. When two functions are conflated, they become identified as a single function bundle sharing all properties of both functions. The same kind of identification is the result of unifying two FDs, as illustrated in Figure 5. This unification corresponds to the conflation of AGENT with SUBJECT. A single FD is created to represent the value of both functions.

Table 1 summarizes how the realization operators of SFG can be translated into FUG. Different types of ordering can be expressed as different pattern configurations, as shown in Table 2.

During a first experiment, approximately 80 systems from the clause rank of the Nigel grammar were translated into FUG. Subsequently, a program was implemented to automatically translate any system network into a functional description. This program verifies the effectiveness of the method outlined here for translating a system network into FUG notation. In practice, it is convenient to translate each rank of a grammar into a separate FD.

Translating the information of individual systems presented very few problems. The most significant problems have been encountered in making sure that the dependencies between different systems are adequately expressed. The methods used in solving these problems are described in section 4.3.

4 How FUG Differs from Systemic Grammar

4.1 Purpose

The primary difference between FUG and SFG is one of purpose. FUG is a method for representing grammars so that they may be used by computational tools, while SFG embodies a particular theoretical model of language. FUG allows a great deal of flexibility; it can easily encode many different types of grammatical information as attributes, since attributes may contain as much information and embedded structure as needed. By leaving many options open in this way, the FUG framework by itself does not attempt to serve as a complete theory of language. Rather, FUG is a vehicle for expressing grammatical information in a way suitable for use by a variety of computational processes. FUG and

	SFG realisation	FUG description
Insert	+ SUBJECT	pattern = (... SUBJECT ...)
Preselect	SUBJECT : Nominative	SUBJECT = [Case = Nominative]
Classify	FINITE ! Singular	FINITE = [Number = Singular]
Lexify	FINITE = has	FINITE = [Lex = has]
Conflate	SUBJECT / AGENT	SUBJECT = <AGENT>

Table 1: Summary of FUG equivalents for SFG Realization Operators.

	REALIZATION OPERATOR	FUG PATTERN
Order	SUBJECT ∧ FINITE	(... SUBJECT FINITE ...)
Partition	FINITE | POLARITY	(... FINITE ... POLARITY ...)
OrderAtFront	$^THEME	(THEME ...)
OrderAtEnd	TOPICAL^$	(... TOPICAL)

Table 2: Realization Operators for order expressed in FUG.

closely related models (e.g., PATR-II, described by Shieber, 1984) have been used to represent grammars originating from several linguistic traditions, including Systemic Grammar, Generalized Phrase Structure Grammar and Lexical Functional Grammar.

4.2 Notational Differences

Some superficial differences of notation exist between systemic grammar and its encoding in FUG. The most notable differences are:

1. **System names are used as attribute labels**, whenever a systemic feature is used as a value in an FD, as noted in Section 3. This use of system names is redundant in systemic grammar, because the normal practice is to give a unique name to every feature in the grammar. In FUG the same value might appear with many different attributes, so it is necessary to specify a label. There is no such thing as an unlabeled feature in FUG.

2. **Order operators are expressed as pattern configurations**, as shown in Table 2. The patterns of FUG provide a general framework for describing order constraints. Several alternative methods have been developed as a means of using these patterns in actual computer programs. I use the pattern notation here because of its clarity; I will not elaborate on the specific details of how patterns are handled computationally.

3. **Encoding of Features and Functions.** Features and functions are both encoded by the values of attributes in FUG. A feature is encoded by an atomic value, and a function is encoded by a value that is an FD, specifying the attributes of the constituent that realizes the function. The attributes corresponding to functions can be precisely identified by their occurrence in constituent patterns. Note that as a matter of convention, the labels for attributes corresponding to systemic functions are written in upper case.

These differences stem from the generality of FUG as a representation for grammatical information.

4.3 Extensions to FUG Motivated by SFG

The translation of systemic grammar into FUG has suggested two extensions to the FUG framework:

1. ***designating attributes*** that correspond to systemic features,
2. distinguishing ***selection conditions*** that correspond to complex input conditions of systems.

The first extension was already illustrated in the examples of Section 3. The second extension will be introduced later in section 4.3.2.

4.3.1 Designated attributes for systemic features

The attributes corresponding to systemic features are designated by using the ':=' notation for unification in place of the usual '=' notation. This special designation is necessary to insure that all of the dependencies between different systemic features are enforced. For example, consider the *IndicativeType* system in the network of Figure 11. Both of the following implications are intended for this network.

1. If a clause has the feature *Indicative*, then it must also have exactly one of the features from the *IndicativeType* system (either *Declarative* or *Interrogative*).
2. If a clause has one of the features from the *IndicativeType* system (either *Declarative* or *Interrogative*), then it must also have the feature *Indicative*.

Now consider the FUG representation of this network, which is shown in Figure 12. If the attributes corresponding to the systemic features were not designated with the ':=' notation, only the first of the two implications would hold. It would be possible to unify a FD containing the attribute *IndicativeType = Declarative* with the alternative of the grammar that contains *MoodType = Imperative,* because there is no attribute with the label *IndicativeType* in that alternative. Remember that when two FDs are unified, any attributes present in only one of the FDs are simply copied to the resulting FD.

A two-way dependency exists between the features of these two systems, but only one of the ways is encoded in the FUG representation. Only some of the attributes in a FD should have this kind of two-way dependency. For instance, the attribute *pattern* = (... SUBJECT FINITE ...) does not imply *MoodType = Indicative,* although it occurs in the same alternative with the attribute *IndicativeType = Declarative*. It is not possible to reliably determine which attributes of a FD should have the two-way dependency, unless those attributes are explicitly marked in some way.

By giving a special designation to the attributes that correspond to systemic features it is possible to enforce both of the desired implications. The following interpretation is given to designated attributes. The presence of an attribute with the ':=' notation implies that any disjunctive FD that contains this attribute must have a valid alternative that contains this attribute. Therefore, the presence of a designated attribute (i.e., IndicativeType) also implies that all other such designated attributes in the surrounding portions of the description (i.e., MoodType) must have the value (i.e., Indicative) that is given in the alternative containing this attribute (IndicativeType). Usually at most one attribute of each alternative of a disjunctive FD will be marked as a designated attribute, and this attribute will correspond to a systemic feature.

4.3.2 Distinguishing selection conditions for complex input conditions of systems

Martin Kay's formulation of FUG [Kay79] provides a natural way to express the dependencies between systems by embedded FDs. For example, the system IndicativeType has input condition Indicative and Indicative is a feature chosen in the system MoodType. In FUG, the FD corresponding to the IndicativeType system is embedded within the alternative of MoodType where Indicative is chosen. The method of embedding, combined with designated attributes, works well for simple input conditions, but it does not provide a solution for input conditions made up from complex boolean expressions.

Consider, as an example, the DoingType system from Nigel, shown in Figure 13. Its input condition is the conjunction of two features. One way to express this input condition in FUG is to use a disjunction having two alternatives, as shown in Figure 14. One alternative corresponds to what happens when the DoingType system is entered. The other alternative corresponds to what happens when the DoingType system is not entered. The alternative containing the DoingType system also includes the attributes of the input condition. The other alternative includes the attributes of the *negated* input condition. Notice that the input condition and its negation must both be stated explicitly, unlike in systemic notation.[3] This method of encoding input conditions also requires a computational treatment of negated attributes. Although such a treatment of negation is possible, it can increase the complexity of unification (see Karttunen, 1984).

It should not be necessary to state the negated input condition explicitly, since it can always be derived automatically from the unnegated condition. It is also rather inefficient to mix the attributes of the input condition with the other attributes of the system. The attributes of the input condition contain exactly the information that is needed to choose between the two alternatives of the disjunction (i.e., to choose whether the system is entered or not). Therefore, it would be more efficient and less verbose to have a notation in which the attributes of the input condition are distinguished from the other attributes of the system, and in which those attributes only need to be stated once.

In order to handle complex input conditions we have developed an extension to FUG that uses the input condition as a selection condition on alternatives of a disjunctive FD. This selection condition is expressed as an FD, but it is a description that has a new kind of interpretation in the context of the entire grammar. When the condition is satisfiable (i.e., when all of its attributes can be unified with the values specified for them) then an alternative containing the FD corresponding to the system is chosen. Otherwise, an alternative that does not contain the FD of the system is chosen. This second alternative may be omitted in many cases. The use of a selection condition for the DoingType system is illustrated in Figure 15. The FD of the selection condition appears to the left of the $\longrightarrow$ symbol, and the FD to be used when the selection condition is satisfiable appears to its right.

4.3.3 Summary of extensions

The two extensions described here may have a rather general application. In both cases there is a way of encoding the same information in terms of standard unification, but it

[3] If the negation of the input condition were not included in the alternative that does not include the description of the system, it would be possible to use this alternative even when the input condition for the system holds. Thus the description of the system would not be used when it should be.

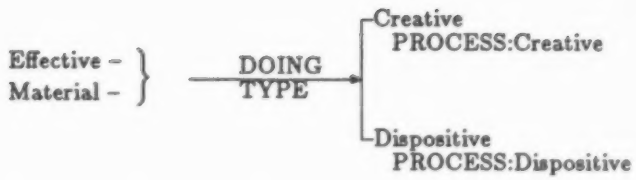


Figure 13: The DoingType System

```
[ Rank = Clause
  ( { [ Agency = Effective
        ProcessType = Material
        { [ DoingType := Creative
            PROCESS = [ Type = Creative ] ]
          [ DoingType := Dispositive
            PROCESS = [ Type = Dispositive ] ] } ]
      [ { [ Agency = NOT Effective ]
          [ ProcessType = NOT Material ] }
        DoingType := none ] } ) ]
```

Figure 14: DoingType system in FUG, using disjunction and negation.

```
[ Rank = Clause
  [ Agency = Effective      ] -> { [ DoingType := Creative
    ProcessType = Material ]     [ PROCESS = [ Type = Creative ] ]
                                 [ DoingType := Dispositive
                                   PROCESS = [ Type = Dispositive ] ] } ]
```

Figure 15: DoingType system in FUG, simplified by using a selection condition.

requires at least doubling the amount of information which encodes that information in the systemic description. Therefore, by adding these extensions, we have not changed what FUG can do, but we have developed a representation that is more perspicuous, less verbose, and computationally more efficient. It appears that these mechanisms may be useful additions to the FUG framework, even when it is not used to directly model a systemic grammar. They suggest that a general treatment of logical implication between FDs might be useful in FUG. One of the most inefficient aspects of the standard unification procedure is the selection of alternatives in disjunctions. The selection condition provides a way of singling out attributes that may be particularly significant in making a selection, so that those attributes can be examined before attempting to unify the rest of the attributes.

Two factors in this study have led to the need for these extensions. The first is, quite naturally, the particular ways in which features are organized in a systemic description. The second is the size of the grammar. Most previous work with FUG has concentrated on the representation of particular phenomena occuring in natural language, but few, if any, attempts have been made to use FUG for a large grammar (i.e., one which approximates an entire language). These extensions may not be important for small grammars, but they seem to become crucial as the size of the grammar grows.

5 Solutions to Problems in Systemic Notation

Matthiessen (in the first part of this report) indicates several problems in developing a more formal algebraic account for phenomena that have been described by systemic linguists. As mentioned in Section 4, FUG offers some flexibility, and this flexibility might be used to augment systemic notation. Here, two of the problems presented by Matthiessen will be addressed: discontinuous constituents and feature propagation.

5.1 Discontinuous Constituents

The problem of discontinuity arises when the constituents that realize a particular function are not sequentially continuous. Matthiessen has indicated that it would be desirable to have a method of simultaneously representing functional continuity and sequential discontinuity. FUG provides such a method, since there is no stipulation that a FD must represent a sequentially continuous unit.

A common type of discontinuity is a split process in a clause, as exemplified by *They called the meeting off*. Figure 16 shows the desired analysis for this clause according to Matthiessen.[4] Figure 17 shows how this analysis can be represented in FUG.

In the FUG representation, a discontinuous constituent (PROCESS) of a CLAUSE contains two subconstituents (PREDICATOR and ADJUNCT), each of which is continuous. Each subconstituent is functionally a part of the PROCESS constituent, but sequentially ordered with respect to other constituents of the CLAUSE, as specified in the pattern,

(... PREDICATOR COMPLEMENT ADJUNCT ...).

The FUG description accounts for both transitivity structure and mood structure. The information about transitivity structure is contained in the description of the PROCESS attribute, and the information about mood structure is contained in the PREDICATOR and

[4]The analysis is adapted from Halliday (1985, p. 186).

	They	called	the meeting	off
Transitivity	AGENT	PRO-	MEDIUM	-CESS
Mood	SUBJECT	PREDICATOR	COMPLEMENT	ADJUNCT

Figure 16: Desired Analysis of Split Process

CLAUSE GRAMMAR contains:

$$\left[\begin{array}{l}
\text{Rank} = \text{Clause} \\
\text{pattern} = (\ldots\ \text{PREDICATOR COMPLEMENT ADJUNCT}\ \ldots) \\
\text{PROCESS} = \left[\begin{array}{l} \text{ProcessType} = \text{Material} \\ \text{PREDICATOR} = [\ \text{Class} = \text{Verb}\ ] \\ \text{ADJUNCT} = [\ \text{Class} = \text{Adverb}\ ] \end{array}\right] \\
\text{PREDICATOR} = \text{<PROCESS PREDICATOR>} \\
\text{ADJUNCT} = \text{<PROCESS ADJUNCT>} \\
\left\{\begin{array}{l} \left[\begin{array}{l} \text{PrimaryTense} = \text{Past} \\ \text{FINITE} = [\ \text{Form} = \text{Past}\ ] \end{array}\right] \\ \left[\begin{array}{l} \text{PrimaryTense} = \text{Present} \\ \ldots \end{array}\right] \\ \left[\begin{array}{l} \text{PrimaryTense} = \text{Future} \\ \ldots \end{array}\right] \end{array}\right\} \\
\left[\begin{array}{l} \text{SecondaryTense} = \text{none} \\ \text{IndicativeType} = \text{Declarative} \end{array}\right] \longrightarrow [\ \text{FINITE} = \text{< PREDICATOR >}\ ]
\end{array}\right]$$

LEXICON contains:

$$\text{"call off"} = \left[\begin{array}{l} \text{ProcessType} = \text{Material} \\ \text{Predicator} = [\ \text{lex} = \text{call}\ ] \\ \text{Adjunct} = [\ \text{lex} = \text{off}\ ] \end{array}\right]$$

$$\text{"call"} = \left[\begin{array}{l} \text{Class} = \text{Verb} \\ \text{lex} = \text{call} \end{array}\right]$$

$$\text{"off"} = \left[\begin{array}{l} \text{Class} = \text{Adverb} \\ \text{lex} = \text{off} \end{array}\right]$$

Figure 17: FUG Description of Split Process

ADJUNCT attributes. In general, FUG is able to handle almost any sort of multifunctional description.

The description also shows how the choice of particular features in the clause grammar adds attributes selectively to the realization of particular subconstituents. When the clause is declarative and the tense is simple (there is no secondary tense), then FINITE is conflated with PREDICATOR. Thus the inflectional form is added to PREDICATOR, but not added to ADJUNCT, although they both function as part of PROCESS.

This solution to the problem of discontinuous constituents also indicates a method for describing *compound lexical items*, or lexical items realized by phrasal units. The lexical item for the compound unit *call-off* has separate attributes for each of the simple lexical items, *call* and *off*. When *call-off* is selected as the realization of PROCESS, its FD is unified with the FD of PROCESS in the clause grammar. As a part of this unification, the attributes for PREDICATOR and ADJUNCT are automatically supplied with appropriate lexical realizations.[5]

5.2 Feature Propagation

Common systemic notation provides one way to specify features that must be selected for a constituent function: preselection. There is no way to specify that two constituents must have some features in common without preselecting those features independently for each constituent. In other words, there is only one type of feature propagation between constituents, the top-down propagation specified by preselection. Because unification is a method for specifying when particular attributes must share the same value, it can be used to produce the effect of more general feature propagation.

An example of feature propagation between different constituents of the same rank is *agreement*. It will be illustrated here by agreement between the subject and finite verb of a clause. At clause rank we would like to specify simply that the finite verb has the same features of person and number as the subject. This relation should hold independently of the particular method that is used to select those features for the subject. A solution of this problem in terms of FUG is shown in Figure 18.

In this solution all attributes that must agree are grouped into a single FD, which is labeled *Agr*. For English, the *Agr* attribute would include the features for person and number with the alternatives:

$$\text{Agr} = \begin{bmatrix} \text{Person} = \{1\ 2\ 3\} \\ \text{Number} = \{\text{singular plural}\} \end{bmatrix}$$

In other languages, agreement might include attributes for gender or case. With this grouping, it is possible to specify agreement at clause rank simultaneously for all of the appropriate attributes.

The grammar of the nominal group specifies the criteria for choosing the values of the agreement attributes, and these values are unified with the *Agr* attribute for the group. For simplex nominal groups, the *Agr* attribute of the group has exactly the same values as the THING function of the group, as indicated by the unification, THING = [Agr = <Agr>]. The path <Agr> denotes the value of the *Agr* attribute for the entire FD of the nominal

[5]For an analysis program it is necessary to provide a more detailed description, so that the composite lexical item *call-off* can be recognized. One way to do this is to have a sense of the word *call* that requires it to be a part of a compound lexical item that contains the adjunct *off*. Other senses of *call* would not require an adjunct. The design of a phrasal lexicon involves many issues that go beyond the scope of this paper.

CLAUSE GRAMMAR contains:

```
[ Rank = Clause
  <FINITE Agr> = <SUBJECT Agr>
  SUBJECT = [ Rank = NounGroup ] ]
```

NOMINAL GROUP GRAMMAR contains:

```
[ Rank = NounGroup
  ( { [ Complexity = Simplex
        THING = [ Agr = <Agr> ] ]
      [ Complexity = Complex
        ( { [ ComplexType = Addition
              Agr = [ Number = Plural ] ]
            [ ComplexType = Alternation
              ... ] } ) ] } ) ]
```

Figure 18: FUG Description of Agreement.

group. For complex nominal groups of the additive type, the *Agr* attribute for the nominal group will have the value [Number = Plural].

When a nominal group realizes the SUBJECT function of a clause, these same values will also be added to the *Agr* attribute for FINITE, because the paths <FINITE Agr> and <SUBJECT Agr> are unified in the clause grammar. Thus the FINITE function must be realized by a constituent having an *Agr* attribute that is compatible with the *Agr* attribute of the SUBJECT.

The solution provided should be applicable to many different types of agreement phenomena. For example, agreement in tag questions could also be handled using the same method. Similar techniques could be used to describe other prosodic phenomena.

These two examples show that translation to FUG can provide a more formal elaboration of ideas that have already been developed informally in systemic theory. Several other notational problems also appear to have a reasonable solution in terms of FUG. For instance, other kinds of interaction across cycles of the grammar are also possible. FUG allows the ranks of systemic grammar to be encoded, using an FD for each rank. However, there are no restrictions on the interdependencies that can be created between ranks, since the framework allows any two attributes of any two FD's to be unified (provided they are compatible).

6 A Grammatical Analysis Program

The grammatical analysis program,[6] which has been under experimental development at USC/ISI, performs the following tasks:

1. segments a sentence into its constituents,
2. assigns functions to each constituent,
3. determines the set of systemic features chosen at each rank (produces a selection expression).

The first two steps represent syntagmatic analysis, and the last step represents paradigmatic analysis. These steps are not necessarily performed in the order given above, rather they are usually performed in an interleaved fashion. Usually particular features need to be recognized before assigning a function to a constituent, and the properties of a function may lead to the determination of additional features.

6.1 How the program works

The type of analysis program most often used with FUG is called a *chart parser*. A good description of how a chart parser works can be found in Winograd's text (Winograd, 1983, pp. 116 – 127). A brief summary is all that can be included here.

A chart is a kind of bookkeeping mechanism, which stores all the information that is known about constituents that have been recognized. It contains *vertices* representing points in the sentence, and *edges* representing constituents (and partially recognized constituents). The vertices are placed between the words of the sentence, and are labeled with indices, beginning with the index 0 to the left of the first word and ending with the index n after the n^{th} word. Each edge connects a pair of vertices and represents a constituent that is composed of the words between those vertices. When a chart is used for FUG, the edges also contain FDs for the constituents that they represent.

The program begins to construct the chart by inserting an edge corresponding to each word of the sentence. A lexicon contains an FD for each sense of a word. These FDs contain attributes for the lexical features of the word. An edge containing a FD for the n^{th} word would have a starting vertex $n-1$ and an ending vertex n. The FD for a word is thus made available to the parser, as it attempts to build larger constituents. In order to build the FD for a constituent, edges from the chart are matched with the descriptions of functions in the grammar. This matching is accomplished by unifying the FD of an edge with the FD for a function. The functions are selected from the grammar, so that they match edges in an order that is consistent with the pattern attributes of the grammar.

The program proceeds (in a bottom-up manner), recognizing groups first, and creating a FD for each group. The FD for each group is then unified with the FD's for functions in the clause grammar. This is the sequence that would be followed for simple clauses. A different sequence would be followed to recognize constructions that result from rank shifting. For example, when a clause occurs as a constituent of a group, then the program would first build an FD for the clause and then unify this FD with the FD for the function of the group that has the attribute *Rank = Clause* in its description.

[6] The program extends the PATR-II system, which has been developed by Shieber et al. at SRI International (Shieber, 1984).

Provided with a method for interpreting the constituent patterns,[7] the chart parsing algorithm and the unification algorithm specify all the work necessary for grammatical analysis of FUG. The chart parsing algorithm determines a strategy for selecting a particular FD from the grammar and the FD for some constituent that has already been recognized. Then, the unification algorithm determines whether these two FDs match, and builds a more complete FD if they can be unified. When the process builds an FD of clause rank covering the entire sentence, then a complete analysis for the sentence has been formed.

6.2 Potential Uses of the Program

The resources offered by a computational analysis program based on systemic functional grammar include:

1. a grammar testing and research tool,
2. text analysis in bulk quantities,
3. a functional approach to natural language processing.

6.2.1 Grammar testing and research tool

Using a grammar in a computational environment requires a degree of precision that may not be necessary in descriptions written for human consumption. It forces the linguist to develop an explicit interpretation of its statements where intuition is replaced by rigour. The grammar becomes more accurate as inconsistencies are detected by a computer program that does not know what the linguist has in mind. A computational program can also reveal examples that are given an analysis other than what was intended by the linguist. The computer cannot correct most mistakes, but it can locate areas of the grammar that need more attention.

I am not proposing that a computer program can replace linguistic intuition, rather that it can help verify and refine linguistic descriptions. It can provide an answer to the question, "Does this system network really work?" When a network gets large it becomes practically impossible for any person to enumerate all possible combinations of feature choices. It may even take a long time for a computer to do this, but computers do not get tired as people do.

This type of analysis program allows a linguist to study aspects of a grammar that have been previously inaccesible. It opens up the possibility of validating the grammar with a much larger body of data than can be done by an individual linguist. Such grammar testing

[7] In order to recognise constituents in an efficient manner, the patterns in an FD must be compiled into an appropriate form. There is a considerable variety of techniques from computer science that can be applied to this task. Kay (1985) gives one method for compiling the patterns into augmented transition networks. Another method involves using a set of phrase structure rules to represent the patterns. This is the method used in the PATR-II parser. The phrase structure rules are used as a processing technique, but they are not significant in the final descriptions produced by the parser.

Using the PATR-II parser, it is not necessary to convert every pattern into explicit phrase structure rules. Some of the patterns may be converted into statements about attributes that mark the start and end of each constituent, corresponding to its endpoints in the chart. For example, in order to encode the pattern (1) the unification (2) can be used.

(... SUBJ FINITE ...) (1)

< SUBJ end > = < FINITE start > (2)

can provide information about the following areas which are described below: coverage, delicacy, consistency, ambiguity, and recoverability.

Coverage: Can the grammar recognize "x"?

The program detects sentences which should be recognized, but which the current expression of the grammar does not include.

Delicacy: Is the set of features adequate for describing "x"?

A superficial grammar will have no features to distinguish sentences that should be given different interpretations. For example, is the position of an adverbial group in a clause functionally significant, or arbitrary? By enumerating the features that are actually assigned to a particular clause by the grammar, an analysis program may show that a feature has been assigned incorrectly, or it may become clear that a feature is missing. Automatic analysis allows an examination of the consequences of treating such variations as either accidental or purposeful.

Consistency: Can all lines of analysis be reconciled?

One of the strengths of systemic grammar is its multifunctional description of language. While desirable, multifunctionality makes it hard to determine whether the grammar is consistent. It is important to know whether the analyses given by the different metafunctions are compatible with each other. If there are incompatibilities, the analysis program will not be able to form a unified description, and it will show some attribute(s) that fail to unify.

Another type of inconsistency can result from the large number of systems in a grammar. The systems are, in general, highly interdependent, so it is not always easy to grasp what effects a change in one system will have on the grammar as a whole. The computational analysis of specific sentences brings out cases where the features of one system do not match the expectations of another.

Ambiguity: How many alternative analyses does the grammar produce for "x"?

It is common for an untested grammar to produce many alternative analyses for some sentences. Only some of these alternatives may be intended by the grammar writer. For example, a fragment of the Nigel grammar produced two analyses for the subject of the clause "Norman is running."

1. SUBJECT = MEDIUM = "Norman"
2. SUBJECT = RANGE = "Norman"

Clearly, only the first interpretation is correct, but the fragment of the grammar that was being tested did not contain enough information to eliminate the second interpretation. The program helps detect such ***unintended analyses.***

Recoverability: Which grammatical functions can be determined for "x"?

Some grammatical functions can be determined only from the context in which a sentence is used. Using the analysis program, it is possible to see which functions cannot be recovered by the grammar. By isolating the difficult cases, it is easier to decide whether the grammar needs to be refined to make better use of the information that it has, or whether the disambiguating information must be found in the extra-sentential context.

6.2.2 Text analysis in bulk quantities

The idea of developing an automated corpus for language studies has interest for many scholars, including some systemic linguists. A program that uses an explicit representation of a grammar offers a qualitative difference from previous work (e.g., syntactic tagging) by developing a more complete account of grammatical features, structures and functions.

6.2.3 Functional approach to natural language processing

In any computer application that requires simulating some degree of understanding human language (e.g., as a method of extracting information from a large database), it is necessary to determine the functions of each constituent in a sequence. Many programs accomplish this task using a limited grammar for specific patterns that occur most often, but it is difficult to extend these grammars to new domains of discourse. A program that has the capability of using any systemic grammar as a basis for language analysis would be able to provide a more general and comprehensive account of grammatical functions than most programs that are currently used for natural language processing.

7 Conclusions

We have shown some ways in which rerepresentation and computational analysis can be used as a testbed for refinement of a systemic grammar. The rerepresentation in the notation of FUG shows that systemic grammars can be made formally explicit, and it is a way of extending the computational techniques that may be applied with the grammar from sentence generation to sentence analysis.

A trial version of the computational analysis program described in Section 6 has been implemented. This program is still under development. Several refinements are necessary to improve its efficiency, and make it available for the practical uses that have been mentioned here.

The comparison of SFG with FUG has indicated some ways in which both frameworks may be enriched. The task of representing a large systemic grammar in FUG has shown where some extensions to FUG are necessary, if it is to be used for grammars approximating those of real human languages. In turn, the solutions to notational problems in terms of FUG have shown some ways in which systemic notation might be enriched. This study has been beneficial for both frameworks; the tools of FUG can be used to improve systemic descriptions by testing and refinement, and applying FUG to systemic grammar has sharpened it as a tool.

The FUG notation also enables a comparison of systemic grammar with other models that use feature-based descriptions, and could make the insights of systemic descriptions available to a wider community of scholars.

Acknowledgements

I would like to acknowledge the role of my colleagues, Bill Mann and Christian Matthiessen, in making this research possible. They introduced me to the Nigel grammar, and have provided a stimulating environment for carrying out this research. Many of the ideas developed in this paper were first formulated during discussions with them. I am also grateful to Susanna Cumming, Lynn Poulton, Fred Swartz, and Christian Matthiessen for helpful comments on an earlier draft of this paper.

References

[Halliday64] Halliday, M. A. K. Syntax and the consumer. In *Report of the 15th Annual Roundtable Meeting of Linguistics and Language Study*. Washington, D.C.: Georgetown University Press. Monograph Series on Languages and Linguistics, 1964. (pp. 11 – 25).

[Halliday85] Halliday, M. A. K. *An Introduction to Functional Grammar*. London: Edward Arnold, 1985.

[Karttunen84] Karttunen, L. Features and values. In *COLING 84: Proceedings of the Tenth International Conference on Computational Linguistics*. Stanford: Association for Computational Linguistics, 1984. (pp. 28 – 33).

[Kay79] Kay, M. Functional grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*. Berkeley: Berkeley Linguistics Society, 1979.

[Kay85] Kay, M. Parsing in functional unification grammar. Chap. 7 in Dowty, D., Kartunnen L., & Zwicky A. (Eds.). *Natural Language Parsing*. Cambridge: Cambridge University Press, 1985.

[Mann & Matthiessen83] Mann, W. C., & Matthiessen, C. *Nigel: a systemic grammar for text generation*. Marina del Rey: University of Southern California, Information Sciences Institute (RR-83-105), 1983. Also in Benson, R., & Greaves, J. (Eds.). *Systemic Perspectives on Discourse: Selected Theoretical Papers from the Ninth International Systemics Workshop*. Norwood, NJ: Ablex, 1985.

[Shieber84] Shieber, S. M. The design of a computer language for linguistic information. In *COLING 84: Proceedings of the Tenth International Conference on Computational Linguistics*. Stanford: Association for Computational Linguistics, 1984. (pp. 362 – 366).

[Winograd83] Winograd, T. *Language as a Cognitive Process, Volume 1: Syntax*. Reading, Massachusetts: Addison-Wesley, 1983.